User’s Guide for the Graphical User Interface

HP 64751 68340 Emulator
HP 64704 Analyzer

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Hewlett-Packard
P.O. Box 2197
1900 Garden of the Gods Road
Colorado Springs, CO 80901-2197, U.S.A.

Printing History

New editions are complete revisions of the manual. The date on the title page changes only when a new edition is published.

A software code may be printed before the date; this indicates the version level of the software product at the time the manual was issued. Many product updates and fixes do not require manual changes, and manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual revisions.

Edition 1	B1442-97000, March 1992
Edition 2	B1442-97001, September 1992
Edition 3	B1442-97002, March 1993
Edition 4	**B1442-97003, December 1993**

Safety, Certification and Warranty

Safety and certification and warranty information can be found at the end of this manual on the pages before the back cover.

68340 Emulation and Analysis

The HP 64751 68340 emulator replaces the microprocessor in your embedded microprocessor system, also called the *target system*, so that you can control execution and view or modify processor and target system resources.

The emulator can be used with the HP 64704 Emulation Bus Analyzer or the HP 64794 Deep Memory Emulation Bus Analyzer which capture 80 channels of emulation processor bus cycle information synchronously with the processor’s clock signal. This analyzer is called the *emulation analyzer*.

With the Emulator, You Can ...

- Plug into 68340 target systems with Pin Grid Array (PGA) sockets.
- Download programs into emulation memory or target system RAM.
- Display or modify the contents of processor registers and memory resources.
- Run programs at clock speeds up to 25 MHz (with active probe boards 64751-66508 and higher — up to 16.78 MHz with boards 64751-66506 and lower), set up software breakpoints, step through programs, and reset the emulation processor.

With the Analyzer, You Can ...

- Trigger the analyzer when a particular bus cycle state is captured. You can also trigger the analyzer after a state has occurred a specified number of times. States are stored relative to the trigger state.
- Qualify which states get stored in the trace.
- Prestore certain states that occur before each qualified store state.
- Trigger the analyzer after a sequence of up to 8 different events have occurred.
- Cause the emulator to stop program execution when the analyzer finds its trigger condition.

With the HP 64700 Card Cage, You Can ...

- Use the RS-422 capability of the serial port and an RS-422 interface card on the host computer (HP 98659 for the HP 9000 Series 300) to provide upload/download rates of up to 230.4K baud.
- Easily upgrade HP 64700 firmware by downloading to flash memory.

With Multiple HP 64700s, You Can ...

- Start and stop up to 16 emulators at the same time (up to 32 if modifications are made).
- Use the analyzer in one HP 64700 to arm (that is, activate) the analyzers in other HP 64700 card cages or to cause emulator execution in other HP 64700 card cages to break.
- Use the HP 64700's BNC connector to trigger an external instrument (for example, a logic analyzer or oscilloscope) when the analyzer finds its trigger condition, or you can allow an external instrument to arm the analyzer or break emulator execution.

With the Graphical User Interface, You Can ...

- Use the emulator and analyzer under an X Window System that supports OSF/Motif interfaces.
- Enter commands using pull-down or pop-up menus.
- Enter, recall, and edit commands using the command line pushbuttons.
- Enter file names, recalled commands, recalled values, etc., using dialog boxes.
- Set breakpoints by pointing the mouse cursor on a line in the mnemonic memory display and clicking.
- Create action keys for commonly used commands or command files.

With the Softkey Interface, You Can ...

- Use the emulator and analyzer with a terminal or terminal emulator.
- Quickly enter commands using softkeys, command recall, and command editing.

In This Book

This book describes the Graphical User Interface and the Softkey Interface when used with the HP 64751 68340 emulator and the HP 64704 analyzer. It is organized into five parts whose chapters are described below.

Part 1. Quick Start Guide

Chapter 1 presents an overview of emulation and analysis and quickly shows you how to use the emulator and analyzer.

Part 2. User's Guide

Chapter 2 shows how to plug the emulator into a target system.
Chapter 3 shows you how to start and exit the HP 64700 interfaces.
Chapter 4 shows you how to enter commands.
Chapter 5 shows how to configure the emulator.
Chapter 6 shows how to use the emulator.
Chapter 7 shows how to use the analyzer.
Chapter 8 shows how to use the Software Performance Measurement Tool (SPMT) with the analyzer.
Chapter 9 shows how to make coordinated measurements.
Chapter 10 shows how to change X resource settings for the Graphical User Interface.

Part 3. Reference

Chapter 11 describes emulator/analyzer interface commands.
Chapter 12 lists the error messages that can occur while using the emulator/analyzer interface.
Chapter 13 lists the emulator specifications and characteristics.

Part 4. Concept Guide

Chapter 14 contains conceptual information on various topics.

Part 5. Installation Guide

Chapter 15 outlines the installation of the Graphical User Interface, and shows you how to start and exit the interface.
Chapter 16 shows you how to install or update emulator firmware.

Contents

Part 2 User's Guide

2 Plugging into a Target System

3 Starting and Exiting HP 64700 Interfaces

4 Entering Commands

10 Setting X Resources

Part 3 Reference

11 Emulator/Analyzer Interface Commands

Part 4 Concept Guide

14 Concepts

Part 5 Installation Guide

15 Installation

Part 1

Quick Start Guide

A one-glance overview of the product and a few task instructions to help you get comfortable.

1

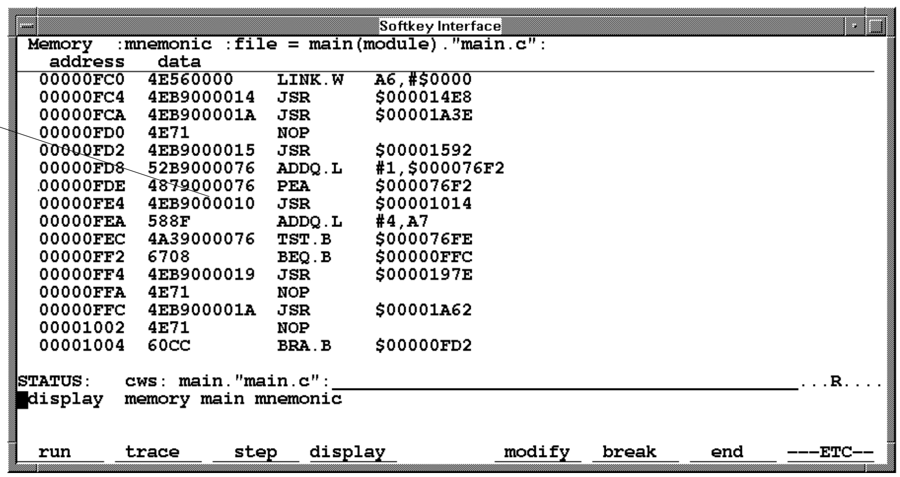
Getting Started

The Emulator/Analyzer Interface — At a Glance

When an X Window System that supports OSF/Motif interfaces is running on the host computer, the emulator/analyzer interface is the Graphical User Interface which provides pull-down and pop-up menus, point and click setting of breakpoints, cut and paste, on-line help, customizable action keys and pop-up recall buffers, etc.

The emulator/analyzer interface can also be the Softkey Interface which is provided for several types of terminals, terminal emulators, and bitmapped displays. When using the Softkey Interface, commands are entered from the keyboard.

The Softkey Interface



Display area. Can show memory, data values, analyzer traces, registers, breakpoints, status, simulated I/O, global symbols, local symbols, pod commands (the emulator's underlying Terminal Interface), error log, or display log. You can use the UP ARROW, DOWN ARROW, PAGE UP, and PAGE DOWN cursor keys to scroll or page up or down the information in the active window.

Status line. Displays the emulator and analyzer status. Also, when error and status messages occur, they are displayed on the status line in addition to being saved in the error log.

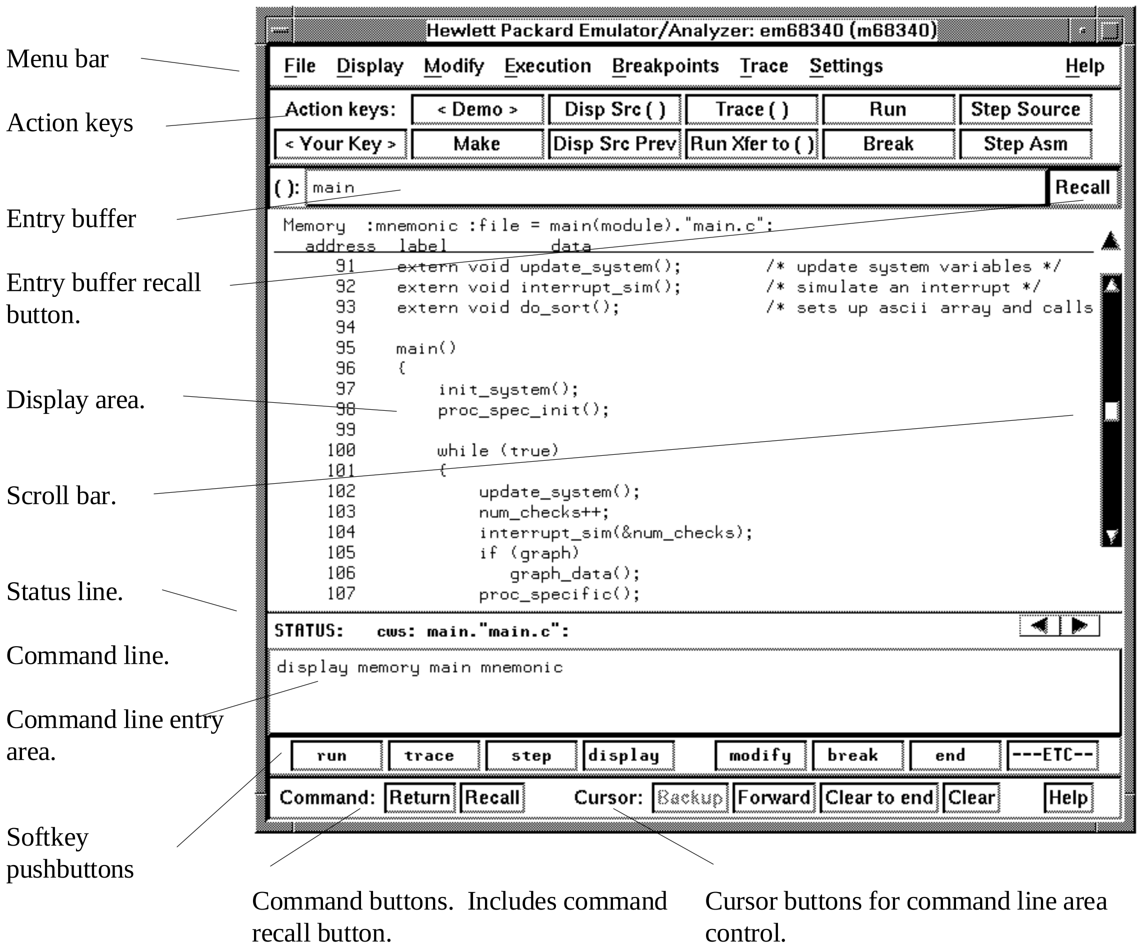
Command line. Commands are entered on the command line by pressing softkeys (or by typing them in) and executed by pressing the Return key. The Tab and Shift-Tab keys allow you to move the cursor on the command line forward or backward. The Clear line key (or CTRL-e) clears from the cursor position to the end of the line. The CTRL-u key clears the whole command line.

Softkey Interface Conventions

Example Softkey Interface commands throughout the manual use the following conventions:

bold	Commands, options, and parts of command syntax.
bold italic	Commands, options, and parts of command syntax which may be entered by pressing softkeys.
normal	User specified parts of a command.
$	Represents the UNIX prompt. Commands which follow the "$" are entered at the UNIX prompt.
<RETURN>	The carriage return key.

The Graphical User Interface



Menu Bar. Provides pulldown menus from which you select commands. When menu items are not applicable, they appear half-bright and do not respond to mouse clicks.

Action Keys. User-defined pushbuttons. You can label these pushbuttons and define the action to be performed.

Entry Buffer. Wherever you see "()" in a pulldown menu, the contents of the entry buffer are used in that command. You can type values into the entry buffer, or you can cut and paste values into the entry buffer from the display area or from the command line entry area. You can also set up action keys to use the contents of the entry buffer.

Entry Buffer Recall Button. Allows you to recall entry buffer values that have been predefined or used in previous commands. When you click on the entry buffer **Recall** button, a dialog box appears that allows you to select values.

Display Area. Can show memory, data values, analyzer traces, registers, breakpoints, status, simulated I/O, global symbols, local symbols, pod commands (the emulator's underlying Terminal Interface), error log, or display log.

Whenever the mouse pointer changes from an arrow to a hand, you can press and hold the *select* mouse button to access popup menus.

Scroll Bar. A "sticky slider" that allows navigation in the display area. Click on the upper and lower arrows to scroll to the top (home) and bottom (end) of the window. Click on the inner arrows to scroll one line. Drag the slider handle up or down to cause continuous scrolling. Click between the inner arrows and the slider handle to page up or page down.

Status Line. Displays the emulator and analyzer status. Also, when error and status messages occur, they are displayed on the status line in addition to being saved in the error log. You can press and hold the *select* mouse button to access the Status Line popup menu.

Command Line. The command line area is similar to the command line in the Softkey Interface; however, the graphical interface lets you use the mouse to enter and edit commands.

- **Command line entry area**. Allows you to enter commands from the command line.
- **Softkey pushbuttons**. Clicking on these pushbuttons, or pressing softkeys, places the command in the command line entry area. You can press and hold the *select* mouse button to access the Command Line popup menu.
- **Command buttons** (includes command recall button). The command **Return** button is the same as pressing the carriage return key — it sends the command in the command line entry area to the emulator/analyzer.

The command **Recall** button allows you to recall previous or predefined commands. When you click on the command **Recall** button, a dialog box appears that allows you to select a command.

- **Cursor buttons for command line area control**. Allow you to move the cursor in the command line entry area forward or backward, clear to the end of the command line, or clear the whole command line entry area.

You can choose not to display the command line area by turning it off. For the most common emulator/analyzer operations, the pulldown menus, popup menus, and action keys provide all the control you need. Choosing menu items that require use of the command line will automatically turn the command line back on.

Graphical User Interface Conventions

Choosing Menu Commands

This chapter uses a shorthand notation for indicating that you should choose a particular menu item. For example, the following instruction

Choose **File→Load→Configuration**

means to first display the **File** pulldown menu, then display the **Load** cascade menu, then select the **Configuration** item from the Load cascade menu.

Based on this explanation, the general rule for interpreting this notation can be stated as follows:

- The leftmost item in bold is the pulldown menu label.
- If there are more than two items, then cascade menus are involved and all items between the first and last item have cascade menus attached.
- The last item on the right is the actual menu choice to be made.

Mouse Button and Keyboard Bindings

Because the Graphical User Interface runs on different kinds of computers, which may have different conventions for mouse buttons and key names, the Graphical User Interface supports different bindings and the customization of bindings.

This manual refers to the mouse buttons using general (or "generic") terms. The following table describes the generic mouse button names and shows the default mouse button bindings.

Mouse Button Bindings and Description

Generic Button Name	**Bindings:** HP 9000	**Sun SPARCsystem**	**Description**
paste	left	left	Paste from the display area to the entry buffer.
command paste	middle[1]	middle[1]	Paste from the entry buffer to the command line text entry area.
select	right	right	Click selects first item in popup menus. Press and hold displays menus.
command select	left	right	Displays pulldown menus.
pushbutton select	left	left	Actuates pushbuttons outside of the display area.

[1] Middle button on three-button mouse. Both buttons on two-button mouse.

The following tables show the default keyboard bindings.

Keyboard Key Bindings

Generic Key Name	HP 9000	Sun SPARCsystem
menu select	extend char	extend char
insert	insert char	insert char
delete	delete char	delete char
left-arrow	left arrow	left arrow
right-arrow	right arrow	right arrow
up-arrow	up arrow	up arrow
down-arrow	down arrow	down arrow
escape	escape	escape
TAB	TAB	TAB

The Getting Started Tutorial

This tutorial gives you step-by-step instructions on how to perform a few basic tasks using the emulator/analyzer interface. The tutorial examples presented in this chapter make the following assumptions:

- The HP 64751 emulator and HP 64704 analyzer are installed into the HP 64700 Card Cage, the HP 64700 is connected to the host computer, and the emulator/analyzer interface software has been installed as outlined in the "Installation" chapter.
- The emulator contains at least 256 Kbytes of emulation memory installed in bank 0 and is plugged into the demo target system.

The Demonstration Program

The demonstration program used in this chapter is a simple environmental control system. The program controls the temperature and humidity of a room requiring accurate environmental control.

Step 1. Start the demo

A demo program and its associated files are provided with the Graphical User Interface.

1 Change to the demo directory.

```
$ cd /usr/hp64000/demo/debug_env/hp64751 <RETURN>
```

Refer to the README file for more information on the demo program.

2 Check that "/usr/hp64000/bin" and "**.**" are in your PATH environment variable. To see the value of PATH:

```
$ echo $PATH <RETURN>
```

3 If the Graphical User Interface software is installed on a different type of computer than the computer you are using, edit the "platformScheme" resource setting in the "Xdefaults.emul" file.

For example, if the Graphical User Interface will be run on a HP 9000 computer and displayed on a Sun SPARCsystem computer, change the platform scheme to "SunOS".

4 Start the emulator/analyzer demo.

```
$ Startemul <logical_emul_name> <RETURN>
```

This script starts the emulator/analyzer interface (with a customized set of action keys), loads a configuration file for the demo program, and then loads the demo program.

The <logical_emul_name> in the command above is the logical emulator name given in the HP 64700 emulator device table file (/usr/hp64000/etc/64700tab.net).

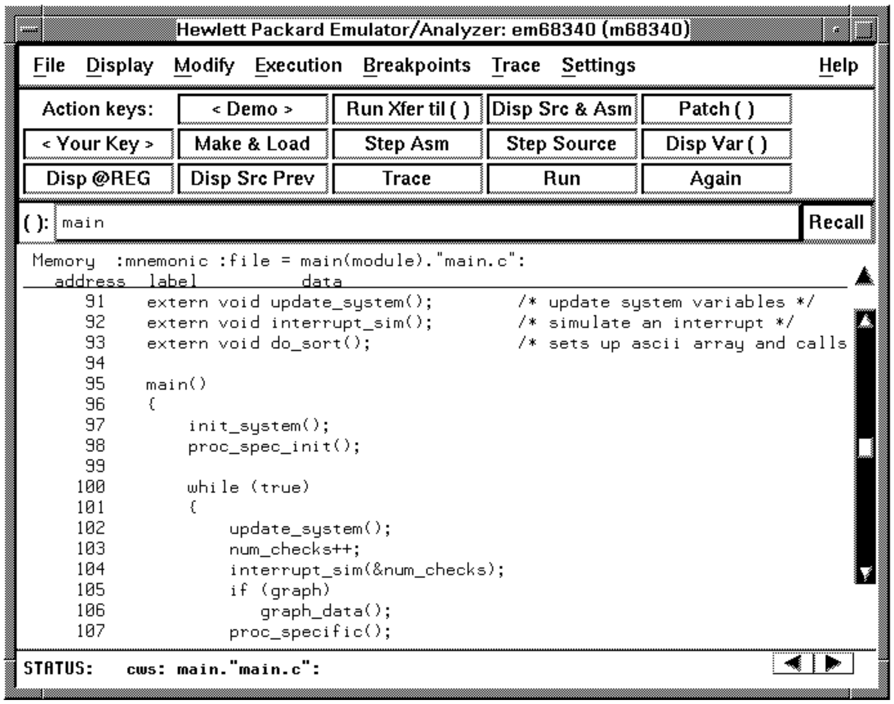
Step 2: Display the program in memory

1 If the symbol "main" is not already in the entry buffer, move the mouse pointer to the entry buffer (notice the flashing I-beam cursor) and type in "main".

2 Choose **Display→Memory→Mnemonic ()**.

Or, using the command line, enter:

`display memory` `main` ***`mnemonic`*** `<RETURN>`



The default display mode settings cause source lines and symbols to appear in displays where appropriate. Notice you can use symbols when specifying expressions. The global symbol "main" is used in the command above to specify the starting address of the memory to be displayed.

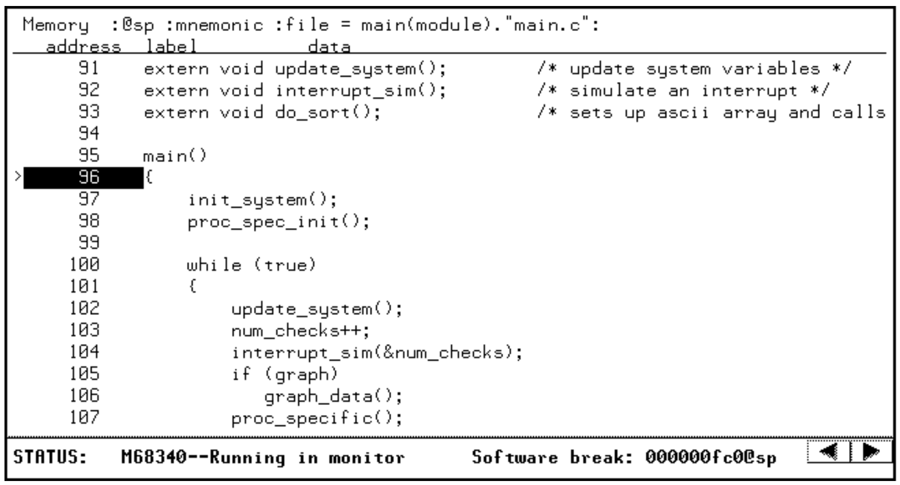
Step 3: Run from the transfer address

The transfer address is the entry address defined by the software development tools and included with the program’s symbol information.

- Click on the **Run Xfer til ()** action key.

Or, using the command line, enter:

run from transfer_address until main <RETURN>



```
 Memory  :@sp :mnemonic :file = main(module)."main.c":
   address  label          data
       91    extern void update_system();        /* update system variables */
       92    extern void interrupt_sim();        /* simulate an interrupt */
       93    extern void do_sort();              /* sets up ascii array and calls
       94
       95    main()
>      96    {
       97        init_system();
       98        proc_spec_init();
       99
      100        while (true)
      101        {
      102            update_system();
      103            num_checks++;
      104            interrupt_sim(&num_checks);
      105            if (graph)
      106               graph_data();
      107            proc_specific();
STATUS:   M68340--Running in monitor      Software break: 000000fc0@sp
```

Notice the message "Software break: <address>" is displayed on the status line and that the emulator is "Running in monitor" (you may have to click the *select* mouse button to remove temporary messages from the status line). When you run until an address, a breakpoint is set at the address before the program is run.

Notice the highlighted bar on the screen; it shows the current program counter.

Step 4: Step high-level source lines

You can step through the program by high-level source lines. The emulator executes as many instructions as are associated with the high-level program source lines.

1 To step a source line from the current program counter, click on the **Step Source** action key.

Or, using the command line, enter:

step source <RETURN>

Notice that the highlighted bar (the current program counter) moves to the next high-level source line.

2 Step into the "init_system" function by continuing to step source lines, either by clicking on the **Step Source** action key, by clicking on the **Again** action key which repeats the previous command, or by entering the **step source** command on the command line.

```
Memory  :@sp :mnemonic :file = init_system(module)."init_system.c":
   address  label          data
      26
      27    void init_val_arr();
      28
      29    void
      30    init_system()
>     31    {    /* FUNCTION init_system() */
      32         /* Initialize the target values for temperature and humidity */
      33         target_temp = 73;
      34         target_humid = 45;
      35
      36         /* Intialize the variables indicating the current environment */
      37         /* conditions */
      38         current_temp = 68;
      39         current_humid = 41;
      40
      41         /* Set starting directions for temp and humid */
      42         temp_dir = up;
```

Step 5: Display the previous mnemonic display

- Click on the **Disp Src Prev** action key.

Or, using the command line, enter:

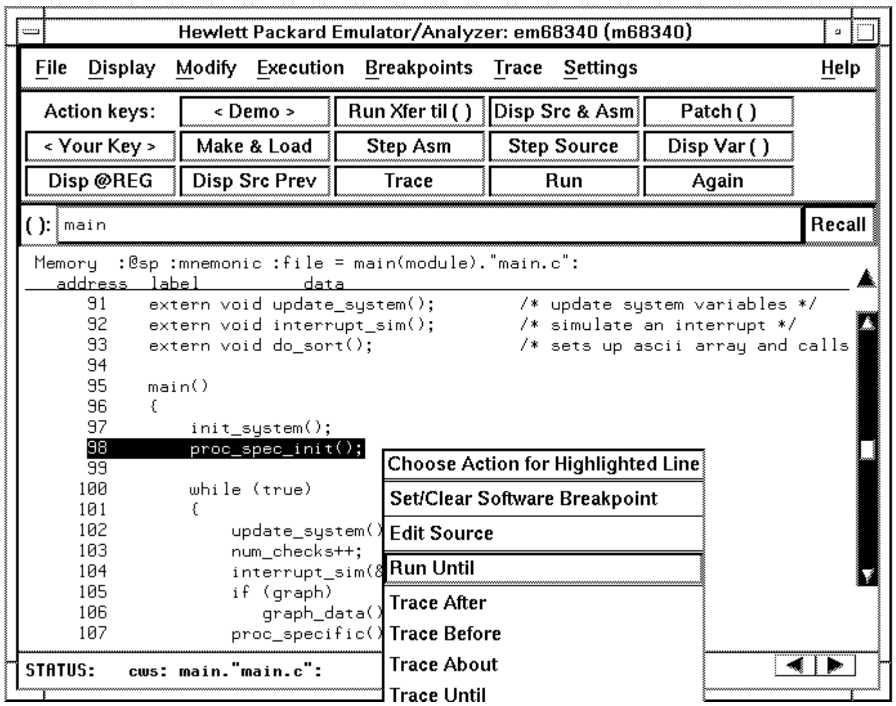
display memory mnemonic previous_display `<RETURN>`

This command is useful, for example, when you have stepped into a function that you do not wish to look at—you can display the previous mnemonic display and run until the source line that follows the function call.

Step 6: Run until an address

When displaying memory in mnemonic format, a selection in the popup menu lets you run from the current program counter address until a specific source line.

- Position the mouse pointer over the line "proc_spec_init();", press and hold the *select* mouse button, and choose **Run Until** from the popup menu.



Or, using the command line, enter:

run until `main."main.c": line 98 <RETURN>`

After the command has executed, notice the highlighted bar indicates the program counter has moved to the specified source line.

Step 7: Display data values

1 Position the mouse pointer over "num_checks" in the source line that reads "num_checks++;" and click the *paste* mouse button (notice "num_checks" is cut and pasted into the entry buffer).

2 Click on the **Disp Var ()** action key.

Or, using the command line, enter:

display data , num_checks ***int32*** <RETURN>

```
Data  :update
  address  label          type        data
 000076F2  |_num_checks  int32                  0
```

The "num_checks" variable is added to the data values display and its value is displayed as a 32-bit integer.

Step 8: Display registers

You can display the contents of the processor registers.

- Choose **Display**→**Registers**→**BASIC**.

Or, using the command line, enter:

display registers <RETURN>

```
Registers
Next PC 00000FCA@sp
 PC 00000FCA  STATUS 2704 < s  z  >     USP 00000000  SSP 00012F94
 D0-D7   00000020 00000020 00008028 0000071C 0000014B 00002AF8 000046F8 00007706
 A0-A7   00007156 FFFFFFFF 00007728 00008020 000077E8 0000F156 00012F94 00012F94
 VBR     00000000  SFC 00   DFC 00
```

Step 9: Step assembly-level instructions

You can step through the program one instruction at a time.

- To step one instruction from the current program counter, click on the **Step Asm** action key.

Or, using the command line, enter:

step <RETURN>

```
Registers
________________________________________________________________________________
Next PC 00000FCA@sp
 PC 00000FCA  STATUS 2704 < s  z  >     USP 00000000  SSP 00012F94
 D0-D7   00000020 00000020 00008028 0000071C 0000014B 00002AF8 000046F8 00007706
 A0-A7   00007156 FFFFFFFF 00007728 00008020 000077E8 0000F156 00012F94 00012F94
 VBR     00000000  SFC 00   DFC 00

Step_PC 00000FCA@sp  JSR      p.proc_spec_init
Next PC 00001A3E@sp
 PC 00001A3E  STATUS 2704 < s  z  >     USP 00000000  SSP 00012F90
 D0-D7   00000020 00000020 00008028 0000071C 0000014B 00002AF8 000046F8 00007706
 A0-A7   00007156 FFFFFFFF 00007728 00008020 000077E8 0000F156 00012F94 00012F90
 VBR     00000000  SFC 00   DFC 00
```

Notice, when registers are displayed, stepping causes the assembly language instruction just executed to be displayed.

Step 10: Trace the program

When the analyzer traces program execution, it looks at the data on the emulation processor's bus and control signals at each clock cycle. The information seen at a particular clock cycle is called a state.

When one of these states matches the "trigger state" you specify, the analyzer stores states in trace memory. When trace memory is filled, the trace is said to be "complete."

1 Click on the **Recall** button to the right of the entry buffer.

A selection dialog box appears. You can select from entry buffer values that have been entered previously or that have been predefined.

2 Click on "main" in the selection dialog box, and click the "OK" pushbutton.

Notice that the value "main" has been returned to the entry buffer.

3 To trigger on the address "main" and store states that occur after the trigger, choose **Trace→After ()**.

Or, using the command line, enter:

trace after main <RETURN>

Notice the message "Emulation trace started" appears on the status line. This shows that the analyzer has begun to look for the trigger state which is the address "main" on the processor's address bus.

4 Run the demo program from its transfer address by choosing **Execution→Run→from Transfer Address**.

Or, using the command line, enter:

run from transfer_address <RETURN>

Notice that now the message on the status line is "Emulation trace complete". This shows the trigger state has been found and the analyzer trace memory has been filled.

5 To view the captured states, choose **Display→Trace**.

Or, using the command line, enter:

display trace <RETURN>

```
Trace List                Offset=0                          More data off screen
Label:      Address           Opcode or Status w/ Source Lines       time count
Base:       symbols               mnemonic w/symbols                  relative
      ##########main.c - line     1 thru    96 #################################
      extern void interrupt_sim();        /* simulate an interrupt */
      extern void do_sort();              /* sets up ascii array and calls combs

      main()
      {
after   prog|main.main     LINK.W   A6,#$0000                         ------------
+001   pr|main+00000002     $0000          supr prgm word rd (ds16)      720     nS
+002   sysstac+00007F94     $0001          supr data long wr (ds16)      720     nS
+003   sysstac+00007F96     $2FF0          supr data word wr (ds16)      720     nS
      ##########main.c - line    97 ############################################
          init_system();
+004   pr|main+00000004    JSR       init.init_system                    680     nS
+005   pr|main+00000006     $0000          supr prgm word rd (ds16)      720     nS
+006   pr|main+00000008     $14E8          supr prgm word rd (ds16)      720     nS
      ##########main.c - line    98 ############################################
```

The default display mode settings cause source lines and symbols to appear in the trace list.

Captured states are numbered in the left-hand column of the trace list. Line 0 always contains the state that caused the analyzer to trigger.

Other columns contain address information, data values, opcode or status information, and time count information.

Step 11: Display memory at an address in a register

1 Click on the **Disp @REG** action key.

Or, using the command line, enter the name of the command file:

```
mematreg <RETURN>
```

A command file dialog box appears (or a prompt appears in the command line).

2 Move the mouse pointer to the dialog box text entry area, type "A7", and click on the "OK" button.

Or, if the prompt is in the command line:

```
A7 <RETURN>
```

```
Memory  :@sp :bytes :blocked :update
  address       data       :hex                                         :ascii
 00012F5C-63    00   00   2A   F8   00   00   2A   F8      . . * .  . . * .
 00012F64-6B    00   00   46   F8   00   01   2F   88      . . F .  . . / .
 00012F6C-73    00   00   10   90   00   00   71   56      . . . .  . . q V
 00012F74-7B    00   00   72   DA   00   00   00   0C      . . r .  . . . .
 00012F7C-83    00   00   80   28   00   00   07   1C      . . . (  . . . .
 00012F84-8B    00   00   01   4B   00   01   2F   94      . . . K  . . / .
 00012F8C-93    00   00   0F   EA   00   00   76   F2      . . . .  . . v .
 00012F94-9B    00   01   2F   F0   00   00   07   10      . . / .  . . . .
 00012F9C-A3    00   00   00   00   00   00   00   00      . . . .  . . . .
 00012FA4-AB    00   00   00   00   00   00   77   E8      . . . .  . . w .
 00012FAC-B3    00   00   00   00   00   00   00   30      . . . .  . . . 0
 00012FB4-BB    00   00   00   02   00   00   00   15      . . . .  . . . .
 00012FBC-C3    00   00   00   01   00   00   00   15      . . . .  . . . .
 00012FC4-CB    00   00   00   00   00   00   00   40      . . . .  . . . @
 00012FCC-D3    00   00   77   28   00   00   00   00      . . w (  . . . .
 00012FD4-DB    00   00   00   C0   00   00   80   28      . . . .  . . . (
 00012FDC-E3    00   00   07   1C   00   00   77   28      . . . .  . . w (
```

Step 12: Patch assembly language code

The **Patch ()** action key lets you patch code in your program.

1 With "main" still in the entry buffer, click on the **Run Xfer til ()** action key.

2 To display memory with assembly-level instructions intermixed with the high-level source lines, click on the **Disp Src & Asm** action key.

```
Memory  :@sp :mnemonic :file = main(module)."main.c":
   address  label          data
       92    extern void interrupt_sim();        /* simulate an interrupt */
       93    extern void do_sort();              /* sets up ascii array and calls
       94
       95    main()
       96    {
> 00000FC0  pr|main.main  4E560000    LINK.W   A6,#$0000
       97        init_system();
  00000FC4               4EB9000014  JSR      init.init_system
       98        proc_spec_init();
  00000FCA               4EB900001A  JSR      p.proc_spec_init
       99
      100        while (true)
  00000FD0               4E71        NOP
      101        {
      102            update_system();
  00000FD2               4EB9000015  JSR      up.update_system
      103            num_checks++;
```

3 Click on the **Patch ()** action key.

A window appears and the **vi** editor is started. Add the line:

```
LINK A6,#1234h
```

Exit out of the editor, saving your changes.

The file you just edited is assembled, and the patch main menu appears. Type "a" and press <RETURN> to apply the patch.

```
Memory  :@sp :mnemonic :file = main(module)."main.c":
   address  label          data
       92    extern void interrupt_sim();        /* simulate an interrupt */
       93    extern void do_sort();              /* sets up ascii array and calls
       94
       95    main()
       96    {
> 00000FC0  pr|main.main  4E561234    LINK.W   A6,#$1234
       97        init_system();
  00000FC4               4EB9000014  JSR      init.init_system
       98        proc_spec_init();
  00000FCA               4EB900001A  JSR      p.proc_spec_init
       99
      100        while (true)
  00000FD0               4E71        NOP
      101        {
      102            update_system();
  00000FD2               4EB9000015  JSR      up.update_system
      103            num_checks++;
```

Notice in the emulator/analyzer interface that the instruction at address "main" has changed.

4 Click on the **Patch ()** action key again.

A window running the **vi** editor again appears, allowing you to modify the patch code that was just created. Modify the line you added previously to:

```
LINK A6,#0
```

Exit out of the editor, saving your changes.

The file you just edited is assembled, and the patch main menu appears. Type "a <RETURN>" to apply the patch.

Notice in the emulator/analyzer interface that the instruction at address "main" has been changed back to what it was originally.

When patching a single address, make sure the new instruction takes up the same number of bytes as the old instruction; otherwise, you may inadvertently modify code that follows.

5 Type "main+4 thru main+15" in the entry buffer.

By entering an address range in the entry buffer (that is, <address> thru <address>) before clicking on the **Patch ()** action key, you can modify a patch template file which allows you to insert as much or as little code as you wish.

6 Click on the **Patch ()** action key again.

A window running the **vi** editor again appears. Suppose you want to patch the demo program so that the proc_spec_init() function is called before the init_system() function. Suppose also that there is memory available at address 8800H. Edit the patch template file as shown below.

```
; PCHS700 Assembly Patch File: PCHmain+4.s
;
; Date : Tue Jun 30 14:06:06 MDT 1992
; Dir  : /users/guest/demo/debug_env/hp64751
; Owner: guest
;
        INCLUDE PCHSINC.s
        ORG main+4
        BRA patch1     ;You may want to change this name!
        ORG 8800h      ;You MUST set this address!
patch1:
; !!!!!!!!! You may need to modify labels and operands of the     !!!!!!!!!
; !!!!!!!!! following code to match your assembler syntax         !!!!!!!!!
; !!!!!!!!! Patching Range: main+4 thru main+15
; !!!!!!!!! Insert new code here !!!!!!!!!
        JSR _proc_spec_init
        JSR _init_system
        BRA main+16    ;You MUST set this address also!
```

Notice that symbols can be used in the patch file. Exit out of the editor, saving your changes.

The file you just edited is assembled, and the patch main menu appears. Type "a" and press <RETURN> to apply the patch.

You can step through the program to view execution of the patch.

Step 13: Exit the emulator/analyzer interface

- To exit the emulator/analyzer interface and release the emulator, choose **File→Exit→Released**.

  Or, using the command line, enter:

  `end release_system` `<RETURN>`

Part 2

User’s Guide

A complete set of task instructions and problem-solving guidelines, with a few basic concepts.

2

Plugging into a Target System

Plugging the Emulator into a Target System

This chapter describes the steps you must perform when connecting the emulator to a target system:

1 Turn OFF power.

2 If the emulator is currently connected to the demo target system or a different target system, unplug the emulator probe.

3 Select the emulator clock source.

4 Plug the emulator probe into the target system.

5 Turn ON power (first the HP 64700, then the target system).

After you plug the emulator into your target system, you must configure the emulator so that it operates properly with your target system (refer to the "Configuring the Emulator" chapter).

CAUTION

Possible Damage to the Emulator Probe. The emulation probe contains devices that are susceptible to damage by static discharge. Therefore, precautionary measures should be taken before handling the microprocessor connector attached to the end of the probe cable to avoid damaging the internal components of the probe by static electricity.

We STRONGLY suggest using a ground strap when handling the emulator probe. A ground strap is provided with the emulator.

Step 1. Turn OFF power

CAUTION

Possible Damage to the Emulator. Make sure target system power is OFF and make sure HP 64700 power is OFF before removing or installing the emulator probe into the target system.

Do not turn HP 64700 power OFF while the emulator is plugged into a target system whose power is ON.

1 If the emulator is currently plugged into a different target system, turn that target system's power OFF.

2 Turn emulator power OFF.

Step 2. Unplug probe from demo target system

1 If the emulator is currently connected to a different target system, unplug the emulator probe; otherwise, disconnect the emulator probe from the demo target system.

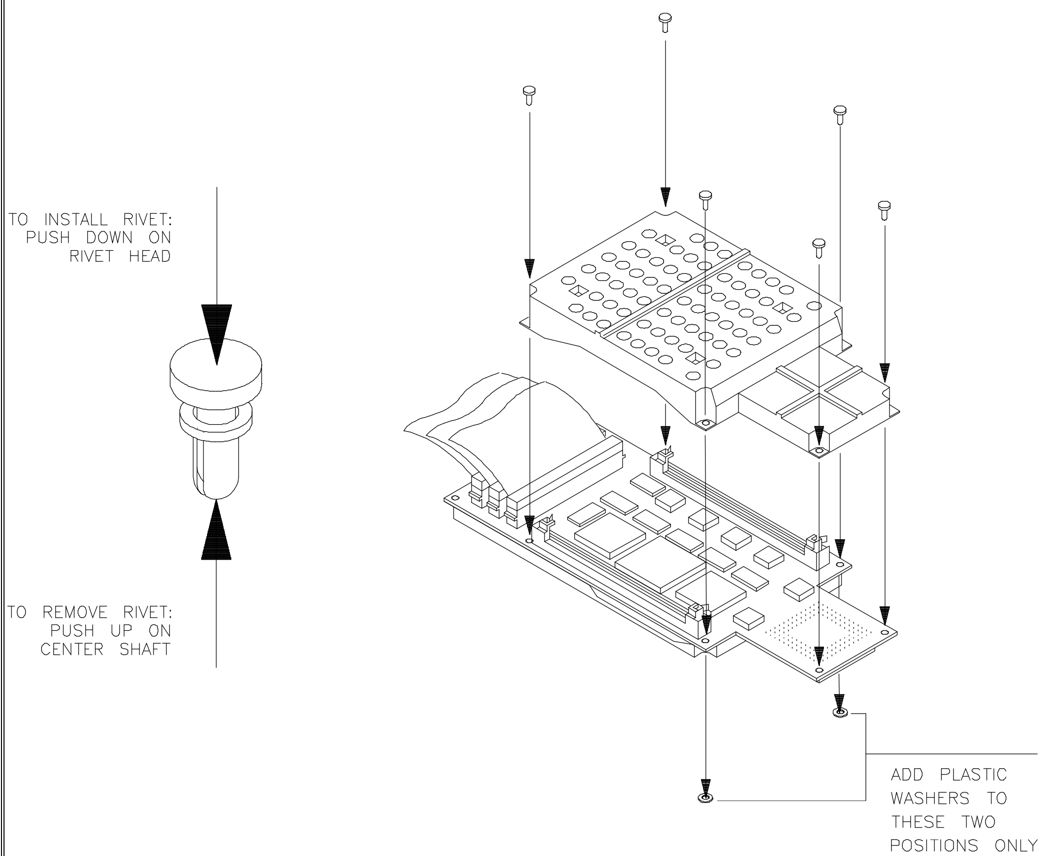
Step 3. Select the emulator clock source

For 64751-66506 and lower numbered active probe printed-circuit boards, the selection of the internal or external clock source is made with the "Micro-processor clock source?" configuration question as described in the "Configuring for Operation with Your Target System" section of the "Configuring the Emulator" chapter.

For 64751-66508 and higher numbered active probe printed-circuit boards, the selection of the internal or external clock source is made by positioning a jumper module on the board.

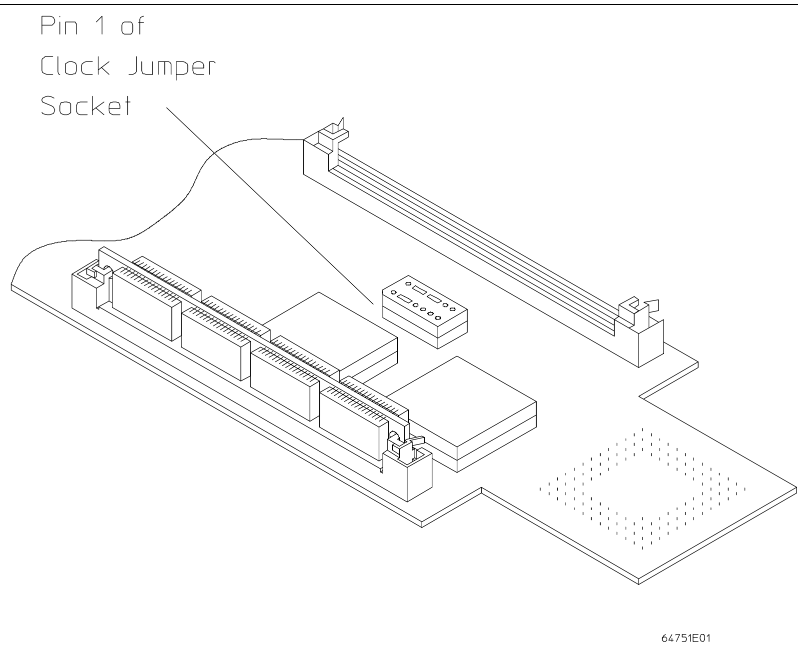
If your active probe board number is 64751-66506 or lower, go on to Step 4; otherwise, perform the following steps.

1 Remove plastic rivets that secure the plastic cover on the top of the emulator probe, and remove the cover.

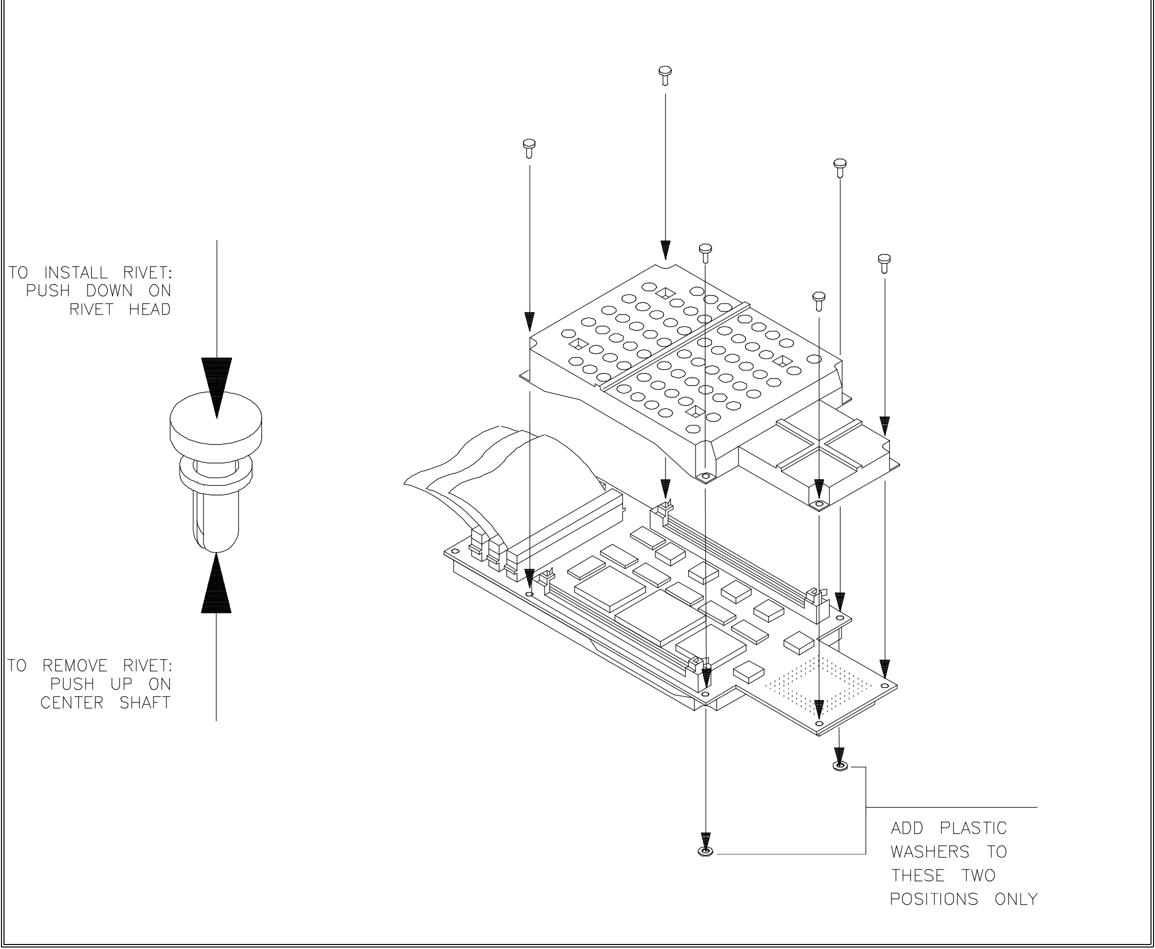


2 To select the 32.768 KHz crystal internal to the emulator, insert the jumper module such that pin 1 of the module aligns with pin 1 of the socket. The target system MUST drive MODCK high (or allow a pullup resistor in the emulator to pull it high) during reset to enable the 68340 VCO and programmable clock mode.

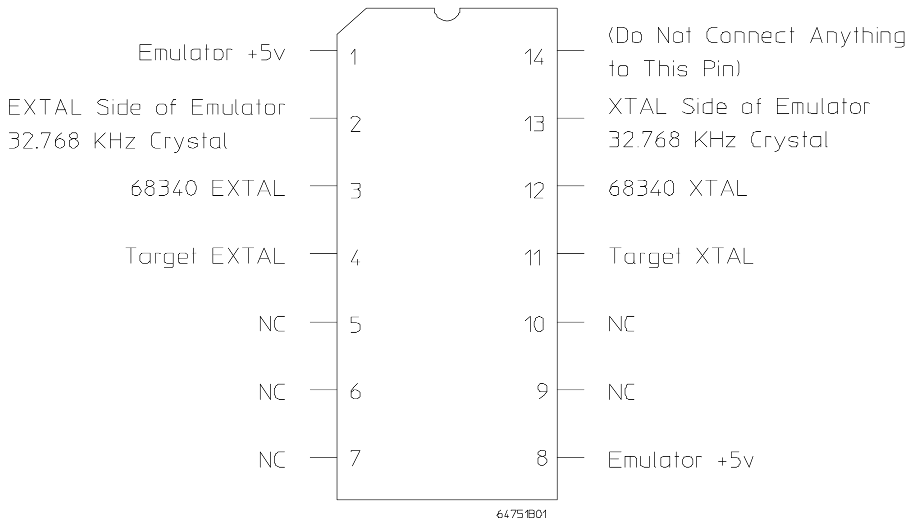
To select an external (target system) TTL oscillator, rotate the jumper module 180 degrees such that pin 8 of the module aligns with pin 1 of the socket. The target system MUST drive MODCK low during reset to enable the 68340 to use the EXTAL signal as the clock source.



3 Replace the plastic cover, and insert new plastic rivets (supplied with the emulator) to secure the cover.



You can also replace the jumper with a prototyping socket on which a crystal and any capacitors or tank circuitry are assembled. (One such prototyping socket is part number 20314-36-455 from Electronic Molding Corp., 96 Mill Street, Woonsocket RI.) The figure below shows the connections that are made to the socket.



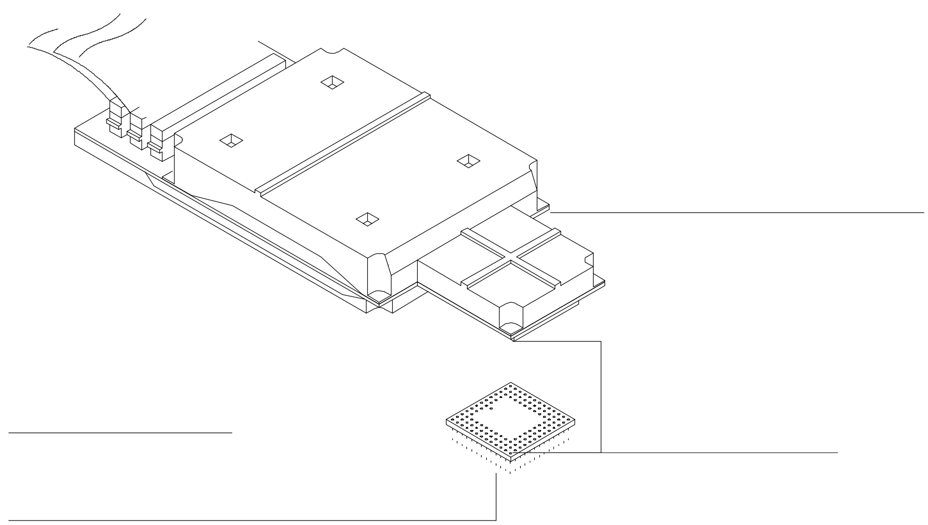
Step 4. Plug the 68340 PGA emulator probe into the target system

CAUTION

Possible Damage to the Emulator Probe. The emulator probe is provided with a pin extender. **Do not use the probe without a pin extender installed**. Replacing a broken pin extender is much less expensive than replacing the emulator probe.

The use of more than one pin extender is discouraged, unless it is necessary for mechanical clearance reasons, because pin extenders cause signal quality degradation.

1 Install the emulator probe into the target system socket. Make sure that pin 1 of the connector aligns with pin 1 of the socket. **Damage to the emulator will result if the probe is incorrectly installed.**



Step 5. Turn ON power

1 Turn emulator power ON.

2 Turn target system power ON.

3

Starting and Exiting HP 64700 Interfaces

Starting and Exiting HP 64700 Interfaces

You can use several types of interfaces to the same emulator at the same time to give yourself different views into the target system.

The strength of the emulator/analyzer interface is that it lets you perform the real-time analysis measurements that are helpful when integrating hardware and software.

The C debugger interface (which is a separate product) lets you view the stack backtrace and high-level data structures, and it lets you use C language expressions and macros. These features are most useful when debugging software.

The Software Performance Analyzer interface (which is also a separate product) lets you make measurements that can help you improve the performance of your software.

These interfaces can operate at the same time with the same emulator. When you perform an action in one of the interfaces, it is reflected in the other interfaces.

Up to 10 interface windows may be started for the same emulator. Only one C debugger interface window and one SPA window are allowed, but you can start multiple emulator/analyzer interface windows.

The tasks associated with starting and exiting HP 64700 interfaces are grouped into the following sections:

- Starting the emulator/analyzer interface.
- Opening other HP 64700 interface windows.
- Exiting HP 64700 interfaces.

Starting the Emulator/Analyzer Interface

Before starting the emulator/analyzer interface, the emulator and interface software must have already been installed as described in the "Installation" chapter.

This section describes how to:

- Start the interface.
- Start the interface using the default configuration.
- Run a command file on interface startup.
- Display the status of emulators defined in the 64700tab.net file.
- Unlock an interface that was left locked by another user.

To start the emulator/analyzer interface

- Use the **emul700 <emul_name>** command.

If **/usr/hp64000/bin** is specified in your PATH environment variable (as shown in the "Installation" chapter), you can start the interface with the **emul700 <emul_name>** command. The "emul_name" is the logical emulator name given in the HP 64700 emulator device table (/usr/hp64000/etc/64700tab.net).

If you are running a window system on your host computer (for example, the X Window System), you can run the interface in up to 10 windows. This capability provides you with several views into the emulation system. For example, you can display memory in one window, registers in another, an analyzer trace in a third, and data in the fourth.

Examples

To start the emulator/analyzer interface for the 68340 emulator:

```
$ emul700 em68340 <RETURN>
```

The "em68340" in the command above is the logical emulator name given in the HP 64700 emulator device table file (/usr/hp64000/etc/64700tab.net).

```
# Blank  lines  and  the  rest of each line after a '#' character are ignored.
# The information in each line must be in the specified order, with  one  line
# for  each  HP series 64700 emulator.  Use blanks or tabs to separate fields.
#
#--------+------------+-----------+------------------------------------------
# Channel|  Logical   | Processor | Remainder of Information for the Channel
#  Type  |   Name     |   Type    | (IP address for LAN connections)
#--------+------------+-----------+------------------------------------------
#  lan:      em68340    m68340     21.17.9.143
serial:      em68340    m68340     myhost /dev/emcom23 OFF 9600 NONE XON 2 8
```

If you're currently running the X Window System, the Graphical User Interface starts; otherwise, the Softkey Interface starts.

The status message shows that the default configuration file has been loaded. If the command is not successful, you will be given an error message and returned to the UNIX prompt. Error messages are described in the "Error Messages" chapter.

To start the interface using the default configuration

- Use the **emul700 -d <emul_name>** command.

In the **emul700 -d <emul_name>** command, the **-d** option says to use the default configuration. The **-d** option is ignored if the interface is already running in another window or on another terminal.

To run a command file on interface startup

- Use the **emul700 -c <cmd_file> <emul_name>** command.

You can cause command files to be run upon starting the interface by using the **-c <cmd_file>** option to the **emul700** command.

Refer to the "Using Command Files" section in the "Entering Commands" chapter for information on creating command files.

Examples

To start the emulator/analyzer interface and run the "startup" command file:

```
$ emul700 -c startup em68340 <RETURN>
```

To display the status of emulators

- Use the **emul700 -l** or **emul700 -lv** command.

The **-l** option of the **emul700** command lists the status of all emulators defined in the 64700tab and 64700tab.net files. If a logical emulator name is included in the command, just the status of that emulator is listed.

You can also use the **-v** option with the **-l** option for a verbose listing of the status information.

Examples

To list, verbosely, the status of the emulator whose logical name is "em68340":

```
$ emul700 -lv em68340 <RETURN>
```

The information may be similar to:

```
em68340 - m68340 running; user = guest
    description:      M68340 emulation, w/internal analysis, 260Kb emul mem
    user interfaces:  xdebug, xemul, xperf, skemul, sktiming
    device channel:   /dev/emcom23
```

Or, the information may be similar to:

```
em68340 - m68340 running; user = guest@myhost
    description:      M68340 emulation w/internal analysis, 260Kb emul mem
    user interfaces:  xdebug, xemul, xperf, skemul, sktiming
    internet address: 21.17.9.143
```

To unlock an interface that was left locked by another user

- Use the **emul700 -U <emul_name>** command.

The **-U** option to the **emul700** command may be used to unlock the emulators whose logical names are specified. This command will fail if there currently is a session in progress.

Examples

To unlock the emulator whose logical name is "em68340":

```
$ emul700 -U em68340 <RETURN>
```

Opening Other HP 64700 Interface Windows

The **File→Emul700** menu lets you open additional emulator/analyzer interface windows or other HP 64700 interface windows if those products have been installed (for example, the software performance analyzer (SPA) interface and the high-level debugger interface).

This section shows you how to:

- Open additional emulator/analyzer interface windows.
- Open the high-level debugger interface window.
- Open the software performance analyzer (SPA) interface window.

To open additional emulator/analyzer windows

- To open additional Graphical User Interface windows, choose **File→Emul700→Emulator/Analyzer** *under Graphic Windows*, or enter the **emul700 <emul_name>** command in another terminal emulation window.

- To open additional conventional Softkey Interface windows, choose **File→Emul700→Emulator/Analyzer** *under Terminal Windows*, or enter the **emul700 -u skemul <emul_name>** command in another terminal emulation window.

You can open additional Graphical User Interface windows, or terminal emulation windows containing the Softkey Interface.

When you open an additional window, the status line will show that this session is joining a session already in progress, and the event log is displayed.

You can enter commands in any window in which the interface is running. When you enter commands in different windows, the command entered in the first window must complete before the command entered in the second window can start. The status lines and the event log displays are updated in all windows.

To open the high-level debugger interface window

- Choose **File→Emul700→High-Level Debugger ...** under "Graphic Windows", or enter the **emul700 -u xdebug <emul_name>** command in another terminal emulation window.

  For information on how to use the high-level debugger interface, refer to the debugger/emulator *User's Guide.*

To open the software performance analyzer (SPA) interface window

- Choose **File→Emul700→Performance Analyzer ...** under "Graphic Windows", or enter the **emul700 -u xperf <emul_name>** command in another terminal emulation window.

  For information on how to use the software performance analyzer, refer to the *Software Performance Analyzer User's Guide.*

Exiting HP 64700 Interfaces

There are several options available when exiting the HP 764700 interfaces. You can simply close one of the open interface windows, or you can exit the debug session by closing all the open windows. When exiting the debug session, you can lock the emulator so that you can continue later, or you can release the emulation system so that others may use it. This section describes how to:

- Close an interface window.
- Exit a debug/emulation session.

To close an interface window

- In the interface window you wish to close, choose **File→Exit→Window**. In the emulator/analyzer interface command line, enter the **end** command with no options.

All other interface windows remain open, and the emulation session continues, unless the window closed is the only one open for the emulation session. In that case, closing the window ends the emulation session, but locks the emulator so that other users cannot access it.

To exit a debug/emulation session

- To exit the interface, save your configuration to a temporary file, and lock the emulator so that it cannot be accessed by other users, choose **File→Exit→Locked**. In the emulator/analyzer interface command line, enter the **end locked** command.

- To exit the interface and release the emulator for access by other users, choose **File→Exit→Released**. In the emulator/analyzer interface command line, enter the **end release_system** command.

If you exit the interface locked, the interface saves the current configuration to a temporary file and locks the emulator to prevent other users from accessing it. When you again start the interface with the **emul700** command, the temporary file is reloaded, and therefore, you return to the configuration you were using when you quit the interface locked.

Also saved when you exit the interface locked are the contents of the entry buffer and command recall buffer. These recall buffer values will be present when you restart the interface.

In contrast, if you end released, you must have saved the current configuration to a configuration file (if the configuration has changed), or the changes will be lost.

4

Entering Commands

Entering Commands

When an X Window System that supports OSF/Motif interfaces is running on the host computer, the emulator/analyzer interface is the Graphical User Interface which provides pull-down and pop-up menus, point and click setting of breakpoints, cut and paste, on-line help, customizable action keys and pop-up recall buffers, etc.

The emulator/analyzer interface also provides the Softkey Interface for several types of terminals, terminal emulators, and bitmapped displays. When using the Softkey Interface, commands are entered from the keyboard.

When using the Graphical User Interface, the *command line* portion of the interface gives you the option of entering commands in the same manner as they are entered in the Softkey Interface. If you are using the Softkey Interface, you can only enter commands from the keyboard using the command line.

The menu commands in the Graphical User Interface are a subset of the commands available when using the command line. While you have a great deal of capability in the menu commands, you have even more in the command line.

This chapter shows you how to enter commands in each type of emulator/analyzer interface. The tasks associated with entering commands are grouped into the following sections:

- Using menus, the entry buffer, and action keys.
- Using the command line with the mouse.
- Using the command line with the keyboard.
- Using command files.
- Using pod commands.
- Forwarding commands to other HP 64700 interfaces.

Using Menus, the Entry Buffer, and Action Keys

This section describes the tasks you perform when using the Graphical User Interface to enter commands. This section describes how to:

- Choose a pulldown menu item using the mouse.
- Choose a pulldown menu item using the keyboard.
- Use the popup menus.
- Use the entry buffer.
- Copy and paste to the entry buffer.
- Use action keys.
- Use dialog boxes.
- Access help information.

To choose a pulldown menu item using the mouse (method 1)

1 Position the mouse pointer over the name of the menu on the menu bar.

2 Press and hold the *command select* mouse button to display the menu.

3 While continuing to hold down the mouse button, move the mouse pointer to the desired menu item. If the menu item has a cascade menu (identified by an arrow on the right edge of the menu button), then continue to hold the mouse button down and move the mouse pointer toward the arrow on the right edge of the menu. The cascade menu will display. Repeat this step for the cascade menu until you find the desired menu item.

4 Release the mouse button to select the menu choice.

If you decide not to select a menu item, simply continue to hold the mouse button down, move the mouse pointer off of the menu, and release the mouse button.

Some menu items have an ellipsis ("...") as part of the menu label. An ellipsis indicates that the menu item will display a dialog or message box when the menu item is chosen.

To choose a pulldown menu item using the mouse (method 2)

1 Position the mouse pointer over the menu name on the menu bar.

2 Click the *command select* mouse button to display the menu.

3 Move the mouse pointer to the desired menu item. If the menu item has a cascade menu (identified by an arrow on the right edge of the menu button), then repeat the previous step and then this step until you find the desired item.

4 Click the mouse button to select the item.

If you decide not to select a menu item, simply move the mouse pointer off of the menu and click the mouse button.

Some menu items have an ellipsis ("...") as part of the menu label. An ellipsis indicates that the menu item will display a dialog or other box when the menu item is chosen.

To choose a pulldown menu item using the keyboard

- To initially display a pulldown menu, press and hold the **menu select** key (for example, the "Extend char" key on a HP 9000 keyboard) and then type the underlined character in the menu label on the menu bar. (For example, "f" for "File". Type the character in lower case only.)

- To move right to another pulldown menu after having initially displayed a menu, press the **right-arrow** key.

- To move left to another pulldown menu after having initially displayed a menu, press the **left-arrow** key.
- To move down one menu item within a menu, press the **down-arrow** key.
- To move up one menu item within a menu, press the **up-arrow** key.
- To choose a menu item, type the character in the menu item label that is underlined. Or, move to the menu item using the arrow keys and then press the **<RETURN>** key on the keyboard.
- To cancel a displayed menu, press the **Escape** key.

The interface supports keyboard mnemonics and the use of the arrow keys to move within or between menus. For each menu or menu item, the underlined character in the menu or menu item label is the keyboard mnemonic character. Notice the keyboard mnemonic is not always the first character of the label. If a menu item has a cascade menu attached to it, then typing the keyboard mnemonic displays the cascade menu.

Some menu items have an ellipsis ("...") as part of the menu label. An ellipsis indicates that the menu item will display a dialog or other box when the menu item is chosen.

Dialog boxes support the use of the keyboard as well. To direct keyboard input to a dialog box, you must position the mouse pointer somewhere inside the boundaries of the dialog box. That is because the interface *keyboard focus policy* is set to *pointer*. That just means that the window containing the mouse pointer receives the keyboard input.

In addition to keyboard mnemonics, you can also specify keyboard accelerators which are keyboard shortcuts for selected menu items. Refer to the "Setting X Resources" chapter and the "Softkey.Input" scheme file for more information about setting the X resources that control defining keyboard accelerators.

To choose popup menu items

1 Move the mouse pointer to the area whose popup menu you wish to access. (If a popup menu is available, the mouse pointer changes from an arrow to a hand.)

2 Press and hold the *select* mouse button.

3 After the popup menu appears (while continuing to hold down the mouse button), move the mouse pointer to the desired menu item.

4 Release the mouse button to select the menu choice.

If you decide not to select a menu item, simply continue to hold the mouse button down, move the mouse pointer off of the menu, and release the mouse button.

The following popup menus are available in the Graphical User Interface:

- Mnemonic Memory Display.
- Breakpoints Display.
- Global Symbols Display.
- Local Symbols Display.
- Status Line.
- Command Line.

To place values into the entry buffer using the keyboard

1 Position the mouse pointer within the text entry area. (An "I-beam" cursor will appear.)

2 Enter the text using the keyboard.

To clear the entry buffer text area from beginning until end, press the **<Ctrl>u** key combination.

To copy-and-paste to the entry buffer

- To copy and paste a discrete text string as determined by the interface, position the mouse pointer over the text to copy and click the *paste* mouse button.

- To specify the exact text to copy to the entry buffer: press and hold the *paste* mouse button; drag the mouse pointer to highlight the text to copy-and-paste; release the *paste* mouse button.

You can copy-and-paste from the display area, the status line, and from the command line entry area.

When you position the pointer and click the mouse button, the interface expands the highlight to include the most complete text string it considers to be discrete. Discrete here means that the interface will stop expanding the highlight in a given direction when it discovers a delimiting character not determined to be part of the string. A common delimiter would, of course, be a space.

When you press and hold the mouse button and drag the pointer to highlight text, the interface copies all highlighted text to the entry buffer when you release the mouse button.

Because the interface displays absolute addresses as hex values, any copied and pasted string that can be interpreted as a hexadecimal value (that is, the string

contains only numbers 0 through 9 and characters "a" through "f") automatically has an "h" appended.

Note If you have multiple Graphical User Interface windows open, a copy-and-paste action in any window causes the text to appear in all entry buffers in all windows. That is because although there are a number of entry buffers being displayed, there is actually only one entry buffer and it is common to all windows. That means you can copy a symbol or an address from one window and then use it in another window.

On a memory display or trace display, a symbol may not be completely displayed because there are too many characters to fit into the width limit for a particular column of the display. To make a symbol usable for copy-and-paste, you can scroll the screen left or right to display all, or at least more, of the characters from the symbol. The interface displays absolute addresses as hex values.

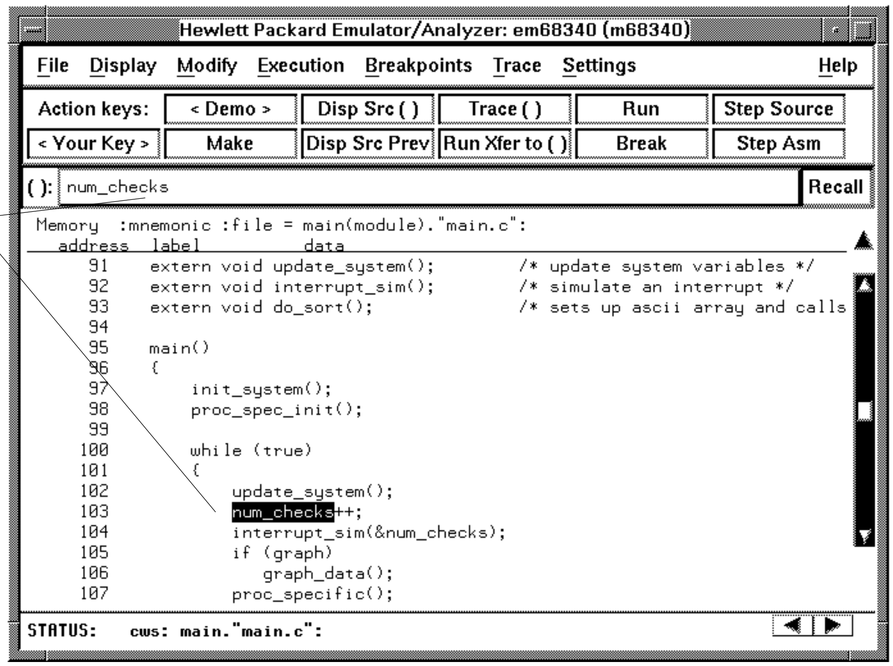
Text pasted into the entry buffer replaces that which is currently there. You cannot use paste to append text to existing text already in the entry buffer.

See "To copy-and-paste from the entry buffer to the command line entry area" for information about pasting the contents of the entry buffer into the command line entry area.

Example

To paste the symbol "num_checks" into the entry buffer from the interface display area, position the mouse pointer over the symbol and then click the paste mouse button.

A mouse click causes the interface to expand the highlight to include the symbol "num_checks" and paste the symbol into the entry buffer.



To recall entry buffer values

- Position the mouse pointer over the **Recall** button just to the right of the entry buffer text area, click the mouse button to bring up the Entry Buffer Recall dialog box, and then choose a string from that dialog box.

The Entry Buffer Recall dialog box contains a list of entries gained during the emulation session as well as any predefined entries present at interface startup.

If you exit the emulation/analysis session with the interface "locked", recall buffer values are saved and will be present when you restart the interface.

You can predefine entries for the Entry Buffer Recall dialog box and define the maximum number of entries by setting X resources (refer to the "Setting X Resources" chapter).

See the following "To use dialog boxes" section for information about using dialog boxes.

To use the entry buffer

1 Place information into the entry buffer (see the previous "To place values into the entry buffer using the keyboard", "To copy-and-paste to the entry buffer", or "To recall entry buffer values" task descriptions).

2 Choose the menu item, or click the action key, that uses the contents of the entry buffer (that is, the menu item or action key that contains "()").

To copy-and-paste from the entry buffer to the command line entry area

1 Place text to be pasted into the command line in the entry buffer text area.

You may do that by:

- Copying the text from the display area using the copy-and-paste feature.
- Enter the text directly by typing it into the entry buffer text area.
- Choose the text from the entry buffer recall dialog box.

2 Position the mouse pointer within the command line text entry area.

3 If necessary, reposition the cursor to the location where you want to paste the text.

4 If necessary, choose the insert or replace mode for the command entry area.

5 Click the *command paste* mouse button to paste the text in the command line entry area at the current cursor position.

The entire contents of the entry buffer are pasted into the command line at the current cursor position.

Although a paste from the display area to the entry buffer affects all displayed entry buffers in all open windows, a paste from the entry buffer to the command line only affects the command line of the window in which you are currently working.

See "To copy-and-paste to the entry buffer" for information about pasting information from the display into the entry buffer.

To use the action keys

1 If the action key uses the contents of the entry buffer, place the desired information in the entry buffer.

2 Position the mouse pointer over the action key and click the action key.

Action keys are user-definable pushbuttons that perform interface or system functions. Action keys can use information from the entry buffer — this makes it possible to create action keys that are more general and flexible.

Several action keys are predefined when you first start the Graphical User Interface. You can use the predefined action keys, but you'll really appreciate them when you define and use your own.

Action keys are defined by setting an X resource. Refer to the chapter "Setting X Resources" for more information about creating action keys.

To use dialog boxes

1 Click on an item in the dialog box list to copy the item to the text entry area.

2 Edit the item in the text entry area (if desired).

3 Click on the "OK" pushbutton to make the selection and close the dialog box, click on the "Apply" pushbutton to make the selection and leave the dialog box open, or click on the "Cancel" pushbutton to cancel the selection and close the dialog box.

The graphical interface uses a number of dialog boxes for selection and recall:

Directory Selection	Selects the working directory. You can change to a previously accessed directory, a predefined directory, or specify a new directory.

File Selection	From the working directory, you can select an existing file name or specify a new file name.
Entry Buffer Recall	You can recall a previously used entry buffer text string, a predefined entry buffer text string, or a newly entered entry buffer string, to the entry buffer text area.
Command Recall	You can recall a previously executed command, a predefined command, or a newly entered command, to the command line.

The dialog boxes share some common properties:

- Most dialog boxes can be left on the screen between uses.
- Dialog boxes can be moved around the screen and do not have to be positioned over the graphical interface window.
- If you iconify the interface window, all dialog boxes are iconified along with the main window.

Except for the File Selection dialog box, predefined entries for each dialog box (and the maximum number of entries) are set via X resources (refer to the "Setting X Resources" chapter).

Examples

To use the File Selection dialog box:

The file filter selects specific files.

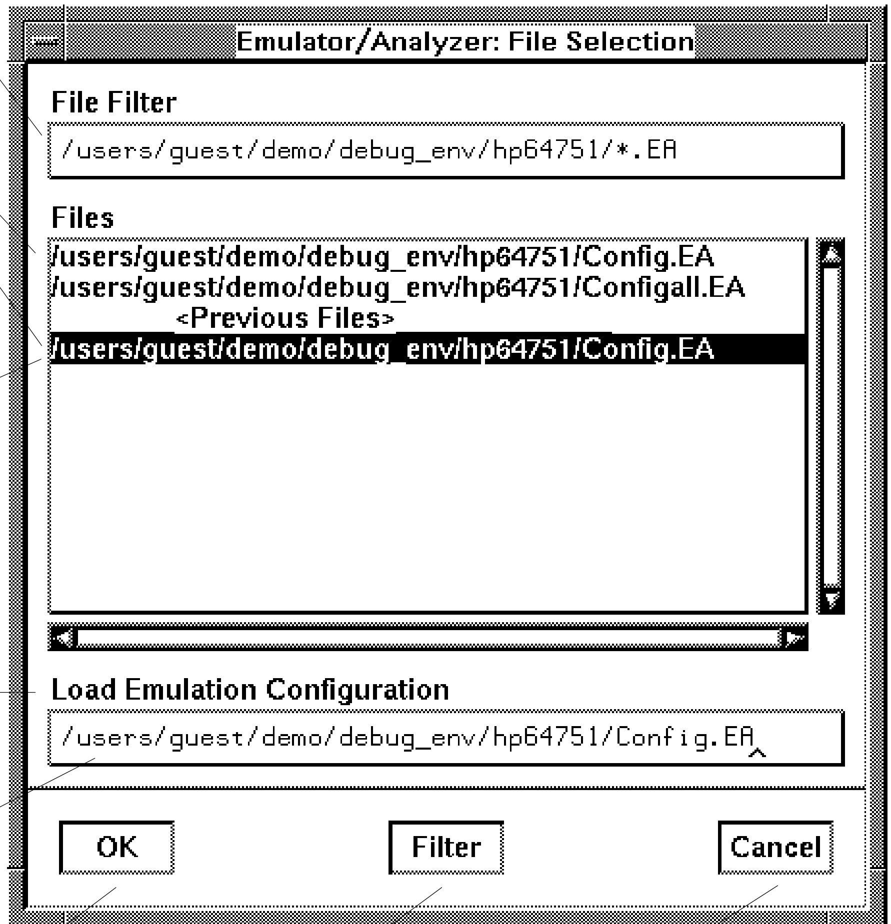
A list of filter-matching files from the current directory.

A list of files previously accessed during the emulation session.

A single click on a file name from either list highlights the file name and copies it to the text area. A double click chooses the file and closes the dialog box.

Label informs you what kind of file selection you are performing.

Text entry area. Text is either copied here from the recall list, or entered directly.



Clicking this button chooses the file name displayed in the text entry area and closes the dialog box.

Entering a new file filter and clicking this button causes a list of files matching the new filter to be read from the directory.

Clicking this button cancels the file selection operation and closes the dialog box.

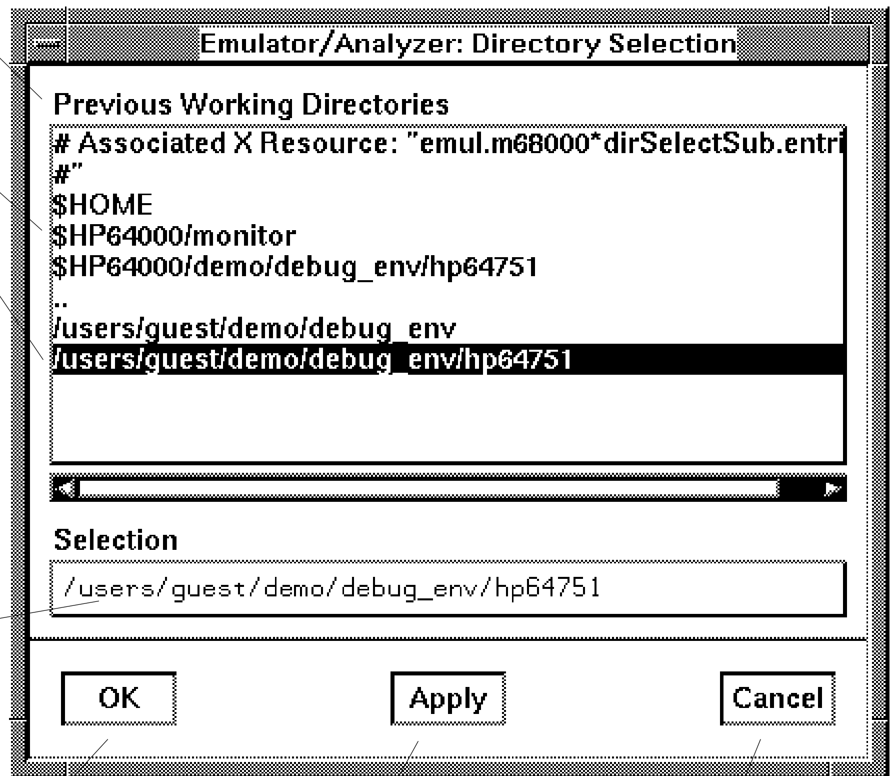
To use the Directory Selection dialog box:

Label informs you of the type of list displayed.

A list of predefined or previously accessed d irectories.

A single click on a directory name from the list highlights the name and copies it to the text area. A double click chooses the directory and closes the dialog box.

Text entry area. Directory name is either copied here from the recall list, or entered directly.



Clicking this button chooses the directory displayed in the text entry area and closes the dialog box.

Clicking this button chooses the directory displayed in the text entry area, but keeps the dialog box on the screen instead of closing it.

Clicking this button cancels the directory selection operation and closes the dialog box.

To access help information

1 Display the Help Index by choosing **Help→General Topic...** or **Help→Command Line...**.

2 Choose a topic of interest from the Help Index.

The Help Index lists topics covering operation of the interface as well other information about the interface. When you choose a topic from the Help Index, the interface displays a window containing the help information. You may leave the window on the screen while you continue using the interface.

Using the Command Line with the Mouse

When using the Graphical User Interface, the *command line* portion of the interface gives you the option of entering commands in the same manner as they are entered in the Softkey Interface. Additionally, the graphical interface makes the softkey labels pushbuttons so commands may be entered using the mouse.

If you are using the Softkey Interface, using the command line with the keyboard is the only way to enter commands.

This section describes how to:

- Turn the command line off/on.
- Enter commands.
- Edit commands.
- Recall commands.
- Display the help window.

To turn the command line on or off

- To turn the command line on or off using the pulldown menu, choose **Settings**→**Command Line**.

- To turn the command line on or off using the status line popup menu: position the mouse pointer within the status line area, press and hold the *select* mouse button, and choose **Command Line Off** from the menu.

- To turn the command line off using the command line entry area popup menu: position the mouse pointer within the entry area, press and hold the *select* mouse button, and choose **Command Line Off** from the menu.

Turns display of the command line area "on" or "off." On means that the command line is displayed and you can use the softkey label pushbuttons, the command return and recall pushbuttons, and the cursor pushbuttons for command line editing.

Off means the command line is not displayed and you use only the pulldown menus and the action keys to control the interface.

The command line area begins just below the status line and continues to the bottom of the emulator/analyzer window. The status line is not part of the command line and continues to be displayed whether the command line is on or off.

Choosing certain pulldown menu items while the command line is off causes the command line to be turned on. That is because the menu item chosen requires some input at the command line that cannot be supplied another way.

To enter a command

1 Build a command using the softkey label pushbuttons by successively positioning the mouse pointer on a pushbutton and clicking the *pushbutton select* mouse button until a complete command is formed.

2 Execute the completed command by clicking the **Return** pushbutton (found near the bottom of the command line in the "Command" group).

Or:

Execute the completed command using the Command Line entry area popup menu: Position the mouse pointer in the command line entry area; press and hold the *select* mouse button until the Command Line popup menu appears; then, choose the **Execute Command** menu item.

You may need to combine pushbutton and keyboard entry to form a complete command.

A complete command is a string of softkey labels and text entered with the keyboard. You know a command is complete when **Return** pushbutton is not halfbright. The interface does not check or act on a command, however, until the command is executed. (In contrast, commands resulting from pulldown menu choices and action keys are supplied with the needed carriage return as part of the command.)

To edit the command line using the command line pushbuttons

- To clear the command line, click the **Clear** pushbutton.
- To clear the command line from the cursor position to the end of the line, click the **Clear to end** pushbutton.
- To move to the right one command word or token, click the **Forward** pushbutton.
- To move to the left one command word or token, click the **Backup** pushbutton.
- To insert characters at the cursor position, press the **insert key** to change to insertion mode, and then type the characters to be inserted.
- To delete characters to the left of the cursor position, press the **<BACKSPACE>** key.

When the cursor arrives at the beginning of a command word or token, the softkey labels change to display the possible choices at that level of the command.

When moving by words left or right, the **Forward** pushbutton becomes halfbright and unresponsive when the cursor reaches the end of the command string. Similarly, the **Backup** pushbutton becomes halfbright and unresponsive when the cursor reaches the beginning of the command.

See "To edit the command line using the mouse and the command line popup menu" and "To edit the command line using the keyboard" for information about additional editing operations you can perform.

To edit the command line using the command line popup menu

- To clear the command line: position the mouse pointer within the Command Line entry area; press and hold the *select* mouse button until the Command Line popup menu appears; choose **Clear Entire Line** from the menu.

- To clear the command line from the cursor position to the end of the line: position the mouse pointer at the place where you want the clear-to-end to start; press and hold the *select* mouse button until the Command Line popup menu appears; choose **Clear to End of Line** from the menu.

- To position the cursor and insert characters at the cursor location: position the mouse pointer in a non-text area of the command line entry area; press and hold the *select* mouse button to display the Command Line popup menu; choose **Position Cursor, Insert Mode** from the menu; type the characters to be inserted.

- To replace characters at the current cursor location: position the mouse pointer in a non-text area of the command line entry area; press and hold the *select* mouse button to display the Command Line popup menu; choose **Position Cursor, Replace Mode** from the menu; type the characters to be inserted.

- To position the cursor and replace characters at the cursor location: position the mouse pointer in a non-text area of the command line entry area; press and hold the *select* mouse button to display the Command Line popup menu; choose **Position Cursor, Replace Mode** from the menu; type the characters to be inserted.

  When the cursor arrives at the beginning of a command word or token, the softkey labels change to display the possible choices at that level of the command.

  See "To edit the command line using the mouse and the command line pushbuttons" and "To edit the command line using the keyboard" for information about additional editing operations you can perform.

To recall commands

1 Click the pushbutton labeled **Recall** in the Command Line to display the dialog box.

2 Choose a command from the buffer list. (You can also enter a command directly into the text entry area of the dialog box.)

Because all command entry methods in the interface — pulldown menus, action keys, and command line entries — are echoed to the command line entry area, the contents of the Command Recall dialog box is not restricted to just commands entered directly into the command line entry area.

The Command Recall dialog box contains a list of interface commands executed during the session as well as any predefined commands present at interface startup.

If you exit the emulation/analysis session with the interface "locked", commands in the recall buffer are saved and will be present when you restart the interface.

You can predefine entries for the Command Recall dialog box and define the maximum number of entries by setting X resources (refer to the "Setting X Resources" chapter).

See "To use dialog boxes" for information about using dialog boxes.

To get help about the command line

- To display the help topic explaining the operation of the command line, press the **Help** pushbutton located near the bottom-right corner of the Command Line area.

Using the Command Line with the Keyboard

When using the command line with the keyboard, you enter commands by pressing softkeys whose labels appear at the bottom of the screen. Softkeys provide for quick command entry, and minimize the possibility of errors.

The command line also provides command completion. You can type the first few characters of a command (enough to uniquely identify the command) and then press <Tab>. The interface completes the command word for you.

Entering commands with the keyboard is easy. However, the interface provides other features that make entering commands even easier. For example, you can:

- Enter multiple commands on one line.
- Recall commands.
- Edit commands.
- Access on-line help information.

To enter multiple commands on one command line

- Separate the commands with semicolons (;).

More than one command may be entered in a single command line if the commands are separated by semicolons (;).

Examples

To reset the emulator and break into the monitor:

reset ; ***break*** <RETURN>

To recall commands

- Press <CTRL>r or <CTRL>b.

The most recent 20 commands you enter are stored in a buffer and may be recalled by pressing <CTRL>r. Pressing <CTRL>b cycles forward through the recall buffer.

Examples

For example, to recall and execute the command prior to the last command:

```
<CTRL>r <CTRL>r <RETURN>
```

To edit commands

- Use the <Left arrow>, <Right arrow>, <Tab>, <Shift><Tab>, <Insert char>, <Back space>, <Delete char>, <Clear line>, and <CTRL>u keys.

The <Left arrow> and <Right arrow> keys move the cursor single spaces to the left or right.

The <Tab> and <Shift><Tab> keys move the cursor to the next or previous word on the command line.

The <Insert char> key enters the insert editing mode and allows characters or command options to be inserted at the cursor location.

The <Back space> key deletes the character to the left of the cursor.

The <Delete char> key deletes the character to the right of the cursor.

The <Clear line> key deletes the characters from the cursor to the end of the line.

The <CTRL>u key erases the command line.

To access on-line help information

- Use the **help** or **?** commands.

To access the command line's on-line help information, type either **help** or **?** on the command line. You will notice a new set of softkeys. By pressing one of these softkeys and <RETURN>, you can display information on that topic.

Examples

To display information on the system commands:

`help` ***`system_commands`*** `<RETURN>`

Or:

`?` ***`system_commands`*** `<RETURN>`

The help information is scrolled on to the screen. If there is more than a screen full of information, you will have to press the space bar to see the next screen full, or the <RETURN> key to see the next line, just as you do with the UNIX **more** command. After all the information on the particular topic has been displayed (or after you press "q" to quit scrolling through information), you are prompted to press <RETURN> to return to the command line.

Using Command Files

You can execute a series of commands that have been stored in a command file. You can create command files by logging commands while using the interface or by using an editor on your host computer.

Once you create a command file, you can execute the file in the emulation environment by typing the name of the file on the command line and pressing <RETURN>.

Command files execute until an end-of-file is found or until a syntax error occurs. You can stop a command file by pressing <CTRL>c or the <Break> key.

This section shows you how to:

- Start logging commands to a command file.
- Stop logging commands to a command file.
- Playback (execute) a command file.

Nesting Command Files

You can nest a maximum of eight levels of command files. Nesting command files means one command file calls another.

Comments in Command Files

Text that follows a pound sign (#), up to the end of the line, is interpreted as a comment.

Using the wait Command

When editing command files, you can insert **wait** commands to pause execution of the command file at certain points.

If you press <CTRL>c to stop execution of a command file while the "wait" command is being executed from the command file, the <CTRL>c will terminate the "wait" command, but will not terminate command file execution. To do this, press <CTRL>c again.

Use the **wait measurement_complete** command after changing the trace depth. By doing this, when you copy or display the trace after changing the trace depth, the new trace states will be available. Otherwise the new states won't be available.

Passing Parameters

Command files provide a convenient method for passing parameters by using a parameter declaration line preceding the commands in the command file. When the command file is called, the system will prompt you for current values of the formal parameters listed.

Parameters are defined as:

> **Passed Parameters -** These are ASCII strings passed to a command file. Any continuous set of ASCII characters can be passed. Spaces separate the parameters.
>
> **Formal Parameters -** These are symbols preceded by an ampersand (**&**), which are the variables of the command file.

The ASCII string passed (passed parameter) will be substituted for the formal parameter when the command file is executed.

The only way to pass a parameter containing a space is to enclose the parameter in double quotes (") or single quotes ('). Thus, to pass the parameter HP 9000 to a command file, you can use either "HP 9000" or 'HP 9000'.

The special parameter **&ArG_lEfT** gets set to all the remaining parameters specified when the command file was invoked. This lets you use variable size parameter lists. If no parameters are left, **&ArG_lEfT** gets set to NULL.

Consider the command file example (named CMDFILE) shown below:

```
PARMS &ADDR &VALUE1
#
# modify a location or list of locations in memory
# and display the result
#
modify memory &ADDR words to &VALUE1 &ArG_lEfT
display memory &ADDR blocked words
```

When you execute CMDFILE, you will be prompted with:

```
Define command file parameter [&ADDR]
```

To pass the parameter, enter the address of the first memory location to be modified. You will then be prompted for **&VALUE1**. If you enter, for example, "0,-1,20, 0ffffh, 4+5*4", the first parameter "0,-1,20," is passed to **&VALUE1** and the remaining parameters "0ffffh," and "4+5*4" are passed to **&ArG_lEfT**.

You can also pass the parameters when you invoke the command file (for example, CMDFILE 1000h 0,-1,20, 0ffffh, 4+5*4).

Other Things to Know About Command Files

You should know the following about using command files:

1 Command files may contain shell variables. Only those shell variables beginning with "$" followed by an identifier will be supported. An identifier is a sequence of letters, digits or underscores beginning with a letter or underscore. The identifier may be enclosed by braces "{ }" or entered directly following the "$" symbol. Braces are required when the identifier is followed by a letter, a digit or an underscore that is not interpreted as part of its name.

   For example, assume a directory named /users/softkeys and the shell variable "S". The value of "S" is "soft". By specifying the directory as /users/${S}keys the correct result is obtained. However, if you attempt to specify the directory as /users/$Skeys, the Softkey Interface looks for the value of the variable "Skeys". This is not the operators intended result. You may not get the intended result unless Skeys is already defined to be "softkeys".

   You can examine the current values of all shell variables defined in your environment with the command "env".

2 Positional shell variables, such as $1, $2, and so on, are not supported. Neither are special shell variables, such as $@, $*, and so on, supported.

3 You can continue command file lines. This is done by avoiding the line feed with a backslash (\). A line terminated by "\" is concatenated with any following lines until a line that does not contain a backslash is found. A line constructed in this manner is recognized and executed as one single command line. If the last line in a command file is terminated by "\", it appears on the command line but is not executed. Normally, the line feed is recognized as the command terminator. The UNIX environment recognizes three quoting

characters for shell commands which are double quotes ("), single quotes (’), and the backslash symbol (\).

For example, the following three lines are treated as a single shell command. The two hidden line feeds are ignored because they are inside the two single quotes (’):

```
!awk ’/$/ { blanks++ }
END { print blanks }
’ an_unix_file
```

To start logging commands to a command file

- Choose **File→Log→Record** and use the dialog box to select a command file name.
- Using the command line, enter the **log_commands to <file>** command.

To stop logging commands to a command file

- Choose **File→Log→Stop**.
- Using the command line, enter the **log_commands off** command.

To playback (execute) a command file

- Choose **File→Log→Playback** and use the dialog box to select the name of the command file you wish to execute.

- Using the command line, enter the name of the command file and press <RETURN>.

If you enter the name of the command file in the command line and the interface cannot find the command file in the current directory, it searches the directories specified in the HP64KPATH environment variable.

To interrupt playback of a command file, press the <CTRL>c key combination. (The mouse pointer must be within the interface window.)

If you press <CTRL>c to stop execution of a command file while the "wait" command is being executed from the command file, the <CTRL>c will terminate the "wait" command, but will not terminate command file execution. To do this, press <CTRL>c again.

Using Pod Commands

Pod commands are Terminal Interface commands. The Terminal Interface is the low-level interface that resides in the firmware of the emulator.

A pod command used in the Graphical User Interface bypasses the interface and goes directly to the emulator. Because some pod commands can cause the interface to become out-of-sync with the emulator, or even cause the interface to terminate abnormally, they must be used with care.

For example, if you change configuration items, the actual state of the emulator will no longer match the internal record the interface keeps about the state of the emulator.

Issuing certain communications-related commands can prevent the interface from communicating with the emulator and cause abnormal termination of the interface.

However, it is sometimes necessary to use pod commands. For example, you must use a pod command to execute the emulator's *performance verification (pv)* routine. Performance verification is an internal self-test procedure for the emulator.

Remember that pod commands can cause trouble for the high-level interface if they are used indiscriminately.

This section shows you how to:

- Display the pod commands screen.
- Use pod commands.

To display the pod commands screen

- Choose **Display→Pod Commands**.

The pod commands screen displays the results of pod (Terminal Interface) commands. To set the interface to use pod commands, choose **Settings→Pod Command Keyboard**.

To use pod commands

- To begin using pod commands, choose **Settings→Pod Command Keyboard**.

- To end using pod commands, click the **suspend** pushbutton softkey.

The **Settings→Pod Command Keyboard** command displays the pod commands screen and activates the keyboard for entering pod command on the command line.

Forwarding Commands to Other HP 64700 Interfaces

To allow the emulator/analyzer interface to run concurrently with other HP 64700 interfaces like the high-level debugger and software performance analyzer, a background "daemon" process is necessary to coordinate actions in the interfaces.

This background process also allows commands to be forwarded from one interface to another. Commands are forwarded using the **forward** command available in the command line. The general syntax is:

```
forward <interface_name> "<command_string>" <RETURN>
```

This section shows you how to:

- Forward commands to the high-level debugger.
- Forward commands to the software performance analyzer.

To forward commands to the high-level debugger

- Enter the **forward debug "<command string>"** command using the command line.

Examples

To send the "Program Run" command to the debugger:

```
forward debug "Program Run" <RETURN>
```

Or, since only the capitalized key is required:

```
forward debug "P R" <RETURN>
```

To forward commands to the software performance analyzer

- Enter the **forward perf "<command string>"** command using the command line.

Examples

To send the "profile" command to the software performance analyzer:

```
forward perf "profile" <RETURN>
```

5

Configuring the Emulator

Configuring the Emulator

This chapter describes how to configure the emulator. You must map memory whenever you use the emulator. When you plug the emulator into a target system, you must configure the emulator so that it operates correctly in the target system. The configuration tasks are grouped into the following sections:

- Using the configuration interface.
- Modifying the general configuration items.
- Reconfiguring the emulator configuration registers.
- Selecting the emulation monitor program.
- Mapping emulation and target system memory.
- Configuring the emulator pod.
- Setting the debug/trace options.

The simulated I/O feature and configuration questions are described in the *Simulated I/O User's Guide.*

The interactive measurement configuration questions are described in the "Making Coordinated Measurements" chapter.

Configuring for Operation in the Target System

After you plug the emulator into a target system and turn on power to the HP 64700, you need to configure the emulator so that it operates properly with your target system.

Before the emulator can operate in your target system, you must:

Map memory. Because the emulator can use target system memory or emulation memory (or both), it is necessary to map ranges of memory so that the emulator knows where to direct its accesses.

You can synchronize emulation memory accesses to the target system in order to more closely imitate target system memory. For example, if emulation memory

replaces slower target system memory that requires wait states, synchronizing emulation memory to the target system causes wait states to be inserted on emulation memory accesses as they would be on target system memory accesses.

Refer to the "Mapping Memory" section later in this chapter.

Select the emulator’s clock source. Generally, you should use the target system clock when plugging the emulator into a target system.

Also, the emulator needs to know the following things:

Is there circuitry in the target system that requires programs to run in real-time? Some emulator commands cause temporary breaks to the monitor state, typically to access microprocessor register values, single-port emulation memory, or target system memory. If the target system requires that programs run in real-time, you must restrict the emulator to real-time runs.

Should the emulator respond to target system interrupts when running in the monitor program? If so, you must use a foreground monitor program since target system interrupts are always ignored during background operation (refer to the "Selecting the Emulation Monitor" section later in this chapter). If it’s not important that the emulator respond to target system interrupts when running in the monitor, you can use the background monitor.

Using the Configuration Interface

This section shows you how to modify, store, and load configurations using the emulator configuration interface.

This section shows you how to:

- Start the configuration interface.
- Modify a configuration section.
- Store a configuration.
- Change the configuration directory context.
- Display the configuration context.
- Access help information.
- Exit the configuration interface.
- Load a configuration.

This chapter describes emulator configuration in general terms. For information about your emulator's specific configuration questions, refer to your emulator *User's Guide*.

To start the configuration interface

- Choose **Modify→Emulator Config...** from the emulator/analyzer interface pulldown menu.

- Using the command line, enter the **modify configuration** command.

  The configuration interface main menu (see example below) is displayed.

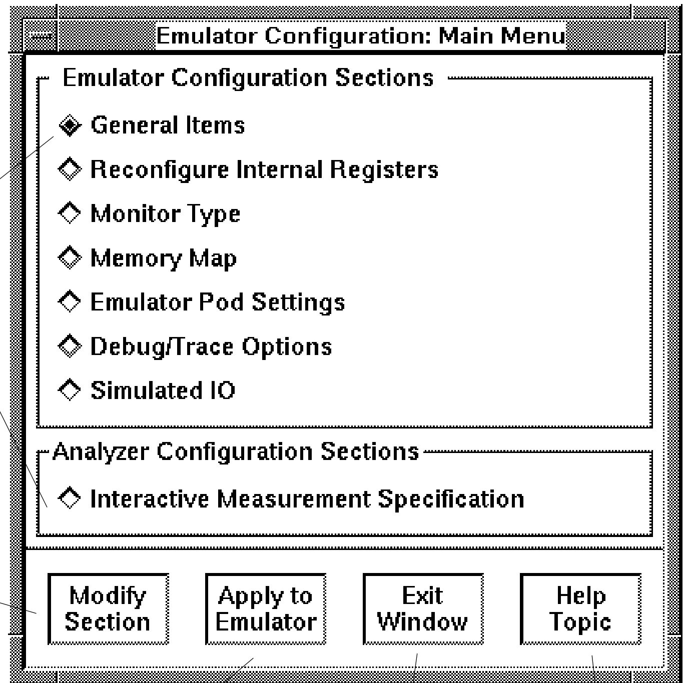
  The configuration sections that are presented depend on the hardware and features of your particular emulator.

  The configuration interface may be left running while you are using the emulator/analyzer interface.

  If you're using the Softkey Interface, you don't get a main menu from which to choose configuration sections; however, the same display area and command line are used to answer the configuration questions.

Examples

The 68340 emulator configuration interface main menu is shown below.



Clicking on one of these lines selects a particular configuration section.

Clicking this button presents the questions for the selected configuration section.

Clicking this button stores the current configuration.

Clicking this button exits the configuration interface.

Clicking this button presents the on-line help.

To modify a configuration section

1 Start the emulator configuration interface.

2 Click on a section name in the configuration interface main menu, and click the "Modify Section" pushbutton.

3 Use the command line to answer the configuration questions.

If you’re using the Softkey Interface:

The configuration questions in the "General Items" section are the first to be asked.

To access the questions in the "Reconfigure Internal Registers" section, answer "yes" to the "Reconfigure internal registers?" question.

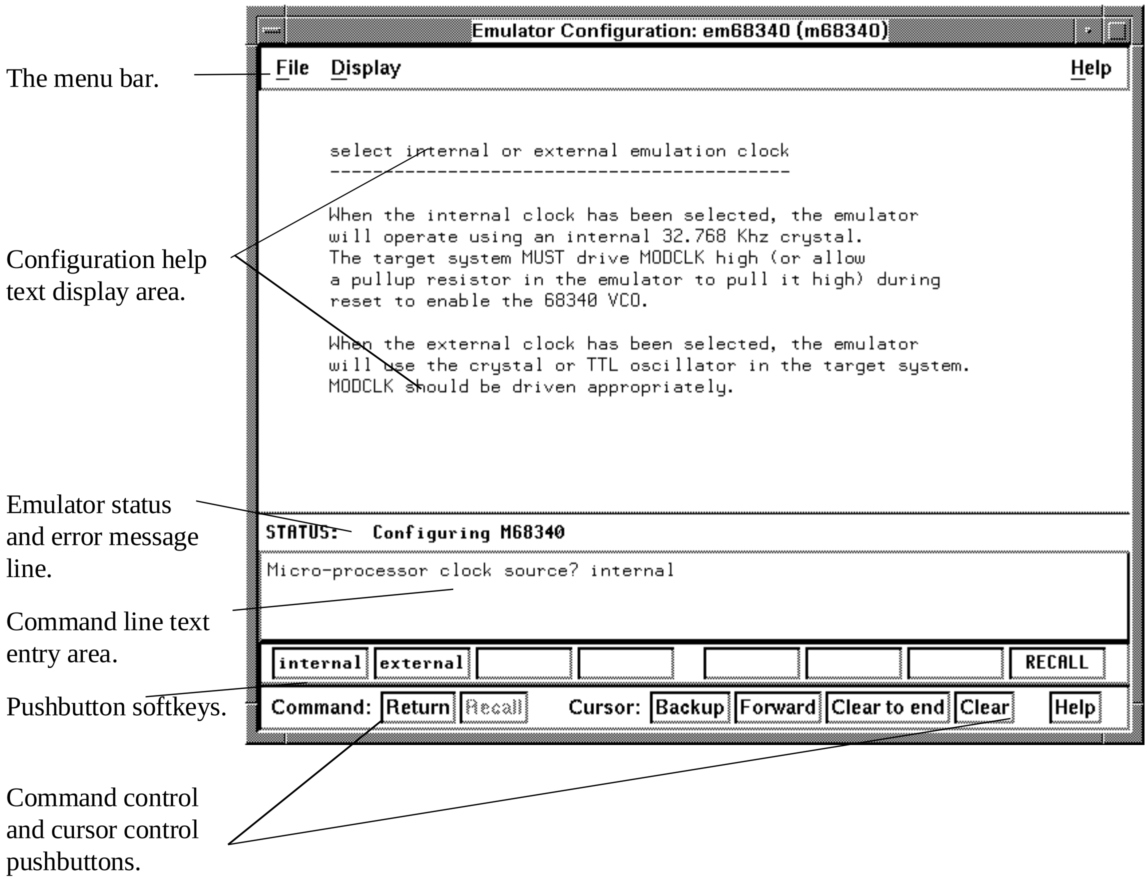
To access the questions in the "Monitor Type" section, answer "yes" to the "Modify memory configuration?" question.

To access the questions in the "Memory Map" section, answer "yes" to the "Modify memory configuration?" question.

To access the questions in the "Emulator Pod Settings" section, answer "yes" to the "Modify emulator pod configuration?" question.

To access the questions in the "Debug/Trace Options" section, answer "yes" to the "Modify debug/trace options?" question.

Each configuration section presents a window similar to the following.



To answer a configuration question, click the softkey pushbutton that has your answer. Or, click on the "Return" command pushbutton to accept the answer that is shown.

When you answer a configuration question, you are normally presented with the next question in the section; however, there are some cases when a carriage return is required, and you can supply it by clicking the "Return" command pushbutton or by pressing the <RETURN> key.

At the last question of a configuration section, you are asked if you wish to return to the main menu. You can click the "next_sec" softkey pushbutton to access the questions in the next configuration section.

To recall a configuration question, click the "RECALL" softkey pushbutton. If you do this at the starting question of a configuration section, you are asked if you want to return to the main menu.

In order for the emulator to recognize any configuration changes, the configuration must be applied to the emulator.

To store a configuration

- When answering the configuration questions, choose **File→Store...** from the pulldown menu, and use the File Selection dialog box to name the configuration file.

- From the configuration interface main menu, click on the "Apply to Emulator" button, and use the File Selection dialog box to name the configuration file.

- If you're using the Softkey Interface, the last configuration question, "Configuration file name?", lets you name the file to which configuration information is stored. If you don't enter a name, configuration information is saved to a temporary file (which is deleted when you exit the interface and release the emulation system).

The file to which the configuration is stored becomes the current configuration file. The emulator only recognizes configuration changes when they are stored or loaded.

When modifying a configuration using the graphical interface, you can store your answers at any time. This is useful for quickly verifying the effect a configuration change has on the emulator.

Configuration information is saved in two files with extensions of ".EA" and ".EB". The file with the ".EA" extension is the "source" copy of the file, and the file with the ".EB" extension is the "binary" or loadable copy of the file.

For more information on how to use dialog boxes, refer to the "To use dialog boxes" description in the "Using Menus, the Entry Buffer, and Action Keys" section of the "Entering Commands" chapter.

To change the configuration directory context

- When answering the configuration questions, choose **File→Directory...** from the pulldown menu, and use the Directory Selection dialog box to specify the new directory.

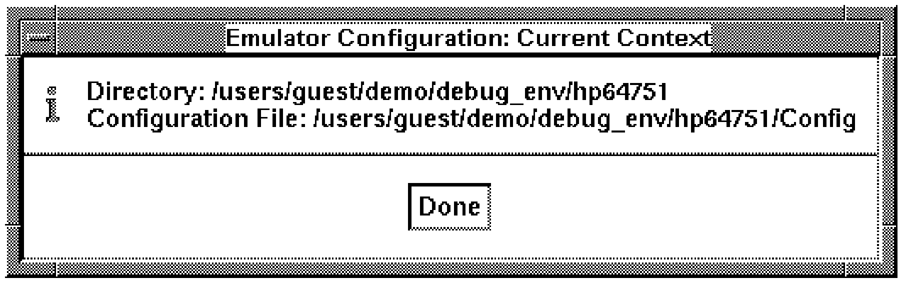
The directory context specifies the directory to which configuration files are stored and from which they are loaded.

For more information on how to use dialog boxes, refer to the "To use dialog boxes" description in the "Using Menus, the Entry Buffer, and Action Keys" section of the "Entering Commands" chapter.

To display the configuration context

- When answering the configuration questions, choose **Display→Context...** from the pulldown menu.

The current directory context and the current configuration files are displayed in a window. Click the "Done" pushbutton when you wish to close the window.



To access help information

- When answering the configuration questions, choose **Help→General Topic...** from the pulldown menu.

- From the configuration interface main menu, click on the "Help Topic" button.

To exit the configuration interface

- When answering the configuration questions, choose **File→Exit...** from the pulldown menu (or type <CTRL>x), and click "Yes" in the confirmation dialog box.

- From the configuration interface main menu, click the "Exit Window" button, and click "Yes" in the confirmation dialog box.

The confirmation dialog box only appears if changes have been made to the current configuration.

When you choose "Yes" from the confirmation dialog box, any modifications made to the configuration which haven't been stored are lost. Choosing "No" from the confirmation dialog box cancels the exit and keeps the emulator configuration interface running.

To load a configuration

- In the emulator/analyzer interface, choose **File→Load→Emulator Config...** from the pulldown menu, and use the File Selection dialog box to specify the configuration file to be loaded.

- Using the command line, enter the **load configuration <FILE>** command.

This command loads previously created and stored configuration files.

Modifying the General Configuration Items

In order to modify the general configuration items, you must first start the configuration interface and access the "General Items" configuration section (refer to the previous "Using the Configuration Interface" section).

This section shows you how to:

- Select the emulator’s clock source.
- Enable/disable entry into the monitor after configuration.
- Restrict to real-time runs.
- Turn OFF the restriction to real-time runs.

To select the emulator’s clock source

For 64751-66508 and higher numbered active probe printed-circuit boards, the selection of the internal or external clock source is made by positioning a jumper module on the board as described in the "Plugging into a Target System" chapter.

For 64751-66506 and lower numbered active probe printed-circuit boards, the selection of the internal or external clock source is made with the "Micro-processor clock source?" configuration question.

- Answer "internal" or "external" to the "Micro-processor clock source?" question.

When you answer "internal", the emulator will use the internal 32.768 KHz crystal. The target system MUST drive MODCLK high (or allow a pullup resistor in the emulator to pull it high) during reset to enable the 68340 voltage-controlled oscillator (VCO).

When you answer "external", the emulator will use the crystal or TTL oscillator in the target system. MODCLK should be driven appropriately.

Generally, you should select the external clock option when using the emulator in-circuit to synchronize the emulator with your target system.

Your target system clock must conform to the specifications for the 68340 microprocessor.

Note that changing the clock source drives the emulator into the reset state. The emulator may later break into the monitor depending on how the "Enter monitor after configuration?" question is answered.

To enable/disable entry into the monitor after configuration

- Answer "yes" or "no" to the "Enter monitor after configuration?".

This question allows you to select whether the emulator will be running in the monitor or held in the reset state on completion of the emulator configuration.

The answer to this configuration question is important in some situations. For example, when you select the external clock and the target system is turned off, do not select reset to monitor. Otherwise, configuration will fail. When you select an external clock source, this question becomes "Enter monitor after configuration (using external clock)?" and the default answer becomes "no".

To restrict to real-time runs

- Answer "yes" to the "Restrict to real-time runs?" question.

While running programs, temporary breaks to the monitor state are not allowed. The emulator refuses the following commands:

- Display or modify registers.
- Display, modify, copy, load, store, or breakpoint commands that access single-port emulation memory or target system memory.

  The emulator contains one 4 Kbyte block of dual-port emulation memory which can be accessed while runs are restricted to real-time. This block of

dual-port emulation memory is reserved for foreground monitor programs when they are used.

- Synchronize SIM registers.

If you want to enter one of these commands, you must first make an explicit break into the monitor using the **break** command.

CAUTION

Target system damage could occur! If your target system circuitry is dependent on constant execution of program code, be aware that the following commands still cause breaks from running programs even when you have restricted the emulator to real-time runs:

- **reset**
- **run**
- **break**
- **step**

Use caution in executing these commands.

To turn OFF the restriction to real-time runs

- Answer "no" to the "Restrict to real-time runs?" question.

Temporary breaks to the monitor while running programs are allowed, and the emulator accepts commands normally.

Reconfiguring the Emulator Configuration Registers

In order to modify the general configuration items, you must first start the configuration interface and access the "Reconfigure Internal Registers" configuration section (refer to the previous "Using the Configuration Interface" section).

This section shows you how to:

- Define values for the emulator configuration registers.

To define values for the emulator configuration registers

- Answer the "Initial value of the configuration copy of <CF_SIM_REGISTER>" questions.

These questions allow you to specify the initial values of the following CF_SIM registers.

CF_MBAR
CF_SIM_MCR
CF_PPARA1
CF_PPARA2
CF_CS0ADDR
CF_CS0MASK
CF_CS1ADDR
CF_CS1MASK
CF_CS2ADDR
CF_CS2MASK
CF_CS3ADDR
CF_CS3MASK

Refer to the "Using the Emulator Configuration Registers" section in the "Using the Emulator" chapter for information on how these registers are used.

Selecting the Emulation Monitor

This section shows you how to:

- Select the background monitor (implemented with the 68340 Background Debug Mode (BDM)).
- Select the foreground monitor program.
- Use a custom foreground monitor program.

When you power up the emulator, or when you initialize it, the background monitor is selected. You can also configure the emulator to use a foreground monitor. Before the background and foreground monitors are described, you should understand the foreground and background emulator modes as well as the function of the emulation monitor.

Background

Background is the emulator mode in which emulation processor execution is suspended.

Foreground

Foreground is the mode in which the emulator executes as if it were a real microprocessor. The emulator is in foreground when it is running user programs or running in a foreground monitor.

Function of the Monitor

The monitor is the interface between the emulation system controller (which accepts and executes emulation commands) and the target system. The monitor uses the emulation microprocessor because that’s the only way to access registers, single-port emulation memory, and target system memory.

When the emulation system controller recognizes that a command requires the monitor, it writes a command code to a communications area and "breaks" emulator execution into the monitor. The monitor reads this command (and any associated parameters), makes the appropriate accesses, places the values in the communication area, and returns emulator execution to its previous state.

Background Monitor

When a background monitor is selected, the Background Debug Mode (BDM) of the 68340 processor is used. The BKPT line is asserted to enter the monitor.

Foreground Monitor

The foreground monitor is an assembly language program that is executed by the 68340 emulation microprocessor in its normal operating mode.

When a foreground monitor is selected, the foreground monitor or downloaded custom monitor is loaded into dual-ported emulation memory and consumes a 4 Kbyte block of the 68340's address range.

The foreground monitor program is included with the interface software as **/usr/hp64000/monitor/fm64751.s**. It can be assembled with the HP AxLS 68000/10/20 Assembler/Linker/Librarian or with the Microtec Research 68000 assembler and linker.

You may customize the foreground monitor if necessary; however, you must maintain the basic communications protocol between the monitor and the emulation system controller. Comments in the monitor program source file detail sections that cannot be changed.

Comparison of Background and Foreground Monitor Programs

Monitor Program Characteristic	Background	Foreground
Takes up processor memory space	No	Yes, 4 Kbytes
Allows the emulator to respond to target system interrupts during monitor execution	No	Yes
Can be customized	No	Yes
Resident in emulator firmware	68340 emulation processor's BDM	Yes, (custom monitor must be assembled, linked, and loaded)

To select the background monitor

1 Access the configuration questions.

2 Answer "yes" to the "Modify memory configuration?" question.

3 Answer "background" to the "Monitor type?" question.

4 Answer "yes" to the "Reset map (change of monitor type requires map reset)?" question.

5 Re-map memory (see the following section on "Mapping Memory").

When a background monitor is selected, the Background Debug Mode (BDM) of the 68340 processor is used. The BKPT line is asserted to enter the monitor.

During background monitor operation, there will be no bus cycle activity except for memory reads and writes that result from memory display or modify commands.

Changing the monitor configuration resets the memory map, so you must re-map memory.

To select the foreground monitor program

1. Access the configuration questions.
2. Answer "yes" to the "Modify memory configuration?" question.
3. Answer "foreground" to the "Monitor type?" question.
4. Answer "yes" to the "Reset map (change of monitor type requires map reset)?" question.
5. Enter 0 through 7 in response to the "Interrupt priority level for default foreground monitor?" question.
6. Enter the base address of the monitor in response to the "Monitor's base address?" question.
7. Answer the "Enable /DSACK interlocking on monitor addresses?" question.
8. Re-map memory (see the following section on "Mapping Memory").
9. Answer "yes" to the "Modify emulator pod configuration?" question.
10. Answer the "Reset value for Supervisor Stack Pointer?" question.
11. Answer the "Reset value for Program Counter?" question.
12. Save the configuration changes.
13. Modify the TRACE exception vector to point to the TRACE_ENTRY symbol in the monitor program so that you can step through the user program.

Selecting the Foreground Monitor

Answering "foreground" to the "Monitor type?" question causes the current memory map to be deleted, and a new map term is added for the monitor program.

The starting address of the monitor block is set by answering the "Monitor's base address?" question, and your response to the "Enable /DSACK interlocking on monitor addresses?" question determines whether the **dsi** (/DSACK interlock) memory attribute is added.

When you select a foreground monitor, the emulator automatically loads the default program, resident in emulator firmware, into dual-ported emulation memory. The foreground monitor is reloaded every time the emulator breaks into the monitor state from the reset state.

Unlike the background monitor, the foreground monitor runs within the same address space as the target program consuming a 4 Kbyte block of the 68340's address range. The foreground monitor can run with target interrupts enabled (see "Selecting the Interrupt Priority Level" below).

The emulator breaks into the foreground monitor by using the emulation processor's background debug mode (BDM) except for single-stepping, which uses the trace exception. The time spent in BDM is approximately 350 microseconds. An exception stack frame of 7 to 13 words will be temporarily pushed onto the user's master and/or interrupt stack(s) during monitor entry.

Selecting the Interrupt Priority Level

The default foreground monitor can be configured to run at a lowered interrupt priority level to allow critical target system interrupts to be processed during monitor execution.

At the point it is safe to lower the interrupt priority level, the foreground monitor will set the interrupt priority mask to the value specified or the interrupt level that was in effect before monitor entry, whichever is greater.

During background monitor operation, all target system interrupts, including level 7 non-maskable interrupts, are blocked.

Modifying this configuration item will reset the processor.

Selecting the Monitor's Base Address

Your response to the "Monitor's base address?" question defines the starting address of the 4 Kbyte block of dual-ported emulation memory. This block of memory is reserved for the foreground monitor. The address must reside on a 4 Kbyte boundary (in other words, an address ending in 000H) and must be specified in hexadecimal. Also, the foreground monitor's base address must have no function code specified.

The current memory map will be deleted, and a new map term is added for the monitor.

Specifying Target Synchronization

If you wish to synchronize monitor cycles to the target system (that is, interlock the emulation and target system /DSACK on accesses to the monitor memory block), answer "yes" to the "Enable /DSACK interlocking on monitor addresses?" question; otherwise, answer "no".

When interlocking is enabled, cycle termination of accesses to foreground monitor memory will not occur until the target system provides a /DSACK. If the monitor is placed in an address range for which the target system does not generate a /DSACK, the emulator will be unable to break into the monitor and a "CPU in wait state" status will result.

When interlocking is disabled, accesses to foreground monitor memory will be terminated by a /DSACK signal generated by the emulator. Any cycle termination signals generated by the target system during monitor memory accesses, including /BERR, will be ignored.

Modifying this configuration item will reset the processor and controls whether the **dsi** (/DSACK Interlock) memory attribute is used in the foreground monitor memory map term.

Re-Mapping Memory

When you configure the emulator for a foreground monitor program, the memory map is reset, and a 4 Kbyte block of emulation memory is automatically mapped for the monitor program. You must re-map other memory ranges before loading user programs.

Modifying the TRACE Exception Vector

In order for single stepping to operate with the foreground monitor, the trace vector in the target system's exception table (VBR plus 24H) must point to the TRACE_ENTRY address in the monitor. This address is equal to the monitor's base address plus 800H in the default foreground monitor.

To use a custom foreground monitor program

1 Edit the monitor program source file to define its base address.

2 Assemble and link the monitor program.

3 Access the configuration questions.

4 Answer "yes" to the "Modify memory configuration?" question.

5 Answer "user_foreground" to the "Monitor type?" question.

6 Answer "yes" to the "Reset map (change of monitor type requires map reset)?" question.

7 Enter the name of the monitor program absolute file in response to the "Monitor filename?" question.

8 Enter the base address of the monitor in response to the "Monitor's base address?" question.

9 Answer the "Enable /DSACK interlocking on monitor addresses?" question.

10 Re-map memory (see the following section on "Mapping Memory").

11 Answer "yes" to the "Modify emulator pod configuration?" question.

12 Answer the "Reset value for Supervisor Stack Pointer?" question.

13 Answer the "Reset value for Program Counter?" question.

14 Save the configuration changes.

15 Modify the TRACE exception vector to point to the TRACE_ENTRY symbol in the monitor program so that you can step through the user program.

Using a custom foreground monitor program is the same as selecting the default foreground monitor, except that the customized monitor program must be assembled, linked, and loaded into emulation memory. Also, the "Interrupt priority level for default foreground monitor?" question is not asked because you can specify the interrupt priority level in the program.

A custom foreground monitor must be assembled and linked starting at the 4 Kbyte boundary specified as the monitor's base address. An ORG statement in the foreground monitor source file defines the base address. Refer to the foreground monitor source provided with the emulator for more information.

The custom foreground monitor is saved in the emulator (until the monitor type is changed) and reloaded every time the emulator breaks into the monitor state from the reset state.

Examples

The following examples of how to set up and use a foreground monitor program assume the HP 64870 or HP B1464 68000/08/10/20/302 Assembler/Linker/Librarian is installed on the host computer.

To copy the foreground monitor program source file:

```
$ cp /usr/hp64000/monitor/fm64751.s . <RETURN>
```

To edit the monitor program source:

```
$ chmod 644 fm64751.s <RETURN>
$ vi fm64751.s <RETURN>
```

The monitor will be loaded at 20000H, so the modify ORG statement near the top of the file to look like this:

```
ORG 020000H  * START MONITOR ON 4K BOUNDARY
```

Notice that the ORG statement is indented from the left margin; if it is not indented, the assembler will interpret the ORG as a label and will generate an error when processing the address portion of the statement.

To assemble and link the monitor program, enter the following commands (which assume that **/usr/hp64000/bin** is defined in the PATH environment variable):

```
$ as68k -L fm64751.s > fm64751.lis <RETURN>

$ ld68k -c fm64751.k -L > fm64751.map <RETURN>
```

Where the “fm64751.k” linker command file is:

```
name fm64751
load fm64751.o
end
```

To configure the emulator to use a foreground monitor program, access the configuration questions, and answer the questions as shown below.

Modify memory configuration? yes
Monitor type? user_foreground
Reset map (change of monitor type requires map reset)? yes
Monitor file name? fm64751
Monitor’s base address? 20000h

Re-map memory for the demo program by entering the following mapper commands:

```
0 thru 6fffh emulation rom <RETURN>
7000h thru 16fffh emulation ram <RETURN>
end <RETURN>
```

Modify emulator pod configuration? yes
Reset value for Supervisor Stack Pointer? 13000h
Reset value for Program Counter? 400h
Configuration file name? fmoncfg

To load the demo program absolute file, enter the following command using the command line:

```
load ecs.x <RETURN>
```

To modify the TRACE exception vector to point to the TRACE_ENTRY label in the monitor program (so that the emulator can single-step), enter the following commands using the command line:

modify memory 24h ***long to*** 20800h <RETURN>

Now, you are ready to use the emulator.

Mapping Memory

Because the emulator can use target system memory or emulation memory (or both), it is necessary to map ranges of memory so that the emulator knows where to direct its accesses.

Up to 7 ranges of memory can be mapped, and the resolution of mapped ranges is 256 bytes bytes (that is, the memory ranges must begin on 256 byte boundaries and must be at least 256 bytes in length).

The emulator contains 4 Kbytes of dual-port emulation memory and provides two slots for additional emulation memory modules:

- HP 64171A 256 Kbyte Memory Modules or HP 64171B 1 Mbyte Memory Modules (0 wait state emulation memory through 16.7 MHz, 1 wait state above 16.7 MHz).
- HP 64172A 256 Kbyte Memory Modules or HP 64172B 1 Mbyte Memory Modules (0 wait state emulation memory through 25 MHz).
- HP 64173A 4 Mbyte Memory Modules (0 wait state emulation memory through 22 MHz, 1 wait state above 22 MHz).

(The 68340 processor is programmed for the correct number of wait states by user code.) If memory modules are mixed, the performance characteristics of the slower module should be used.

Emulation memory is made available to the mapper in blocks. When you map an address range to emulation memory, at least one block is assigned to the range. When a block of emulation memory is assigned to a range, it is no longer available, even though part of the block may be unused.

Emulation memory in bank 0 of the emulator probe is divided into 4 equal blocks, and memory in bank 1 is divided into 2 equal blocks. The 4 Kbyte block of dual-port emulation memory is 1 block.

When you map ranges of emulation memory, blocks are allocated so as to leave the greatest amount of emulation memory available. For example, if you map the range 0 through 0FFH as emulation memory, the 4 Kbyte block of dual-port memory is used if possible; if that block has already been used, the next smallest available block is used.

You should map all memory ranges used by your programs before loading programs into memory.

Using Emulation Memory to Substitute for 8-Bit Memory

Emulation memory is 16-bit wide memory. However, you can use emulation memory to substitute for 8-bit memory by using one of the chip selects and generating internal DSACKx signals for an 8-bit port. You must place the appropriate values into the emulator configuration (CF_SIM) versions of the chip select address and mask registers so that emulation memory is accessed correctly.

Using Chip Selects to Access Emulation Memory

When using chip selects to access emulation memory, the DSACKx signals can be generated internally or externally.

If the DSACKx signals are generated externally (as defined by the **CF_CS**X**MASK** register), emulation memory must be interlocked with the target system (use the **dsi** attribute when mapping the emulation memory range); otherwise, there will be no DSACKx response.

Fast Termination Mode

Emulation memory does not support the fast termination mode (-1 wait state) that can be defined in the chip select registers. If a chip select is programmed for this mode, it will override the mapper and force access to the target system.

External DMA Access to Emulation Memory

External direct memory access (DMA) to emulation memory is not permitted.

The HP 64751 emulator supports operation of the two 68340 on-chip DMA channels in both single- and dual-address modes. Dual-address transfers can access emulation memory; single-address transfers must be between peripherals and memory in the target system only.

To map memory ranges

1 Access the configuration questions.

2 Answer "yes" to the "Modify memory configuration?" question.

3 Enter the address range, memory type, and optionally an attribute for the memory range.

You can characterize memory ranges as emulation RAM, emulation ROM, target system RAM, target system ROM, or as guarded memory.

Guarded memory accesses will cause emulator execution to break into the monitor program.

Writes to locations characterized as ROM will cause emulator execution to break into the monitor program if the "Break processor on write to ROM?" trace/debug configuration option is enabled.

Even though execution breaks into the monitor, the memory location is modified if it’s in emulation ROM or target system RAM mapped as ROM.

The attributes can be:

dp Dual-port emulation memory.

One emulation memory range, up to 4 Kbytes in length, can be given the **dp** attribute. The **dp** attribute specifies that the range be mapped to the 4 Kbyte block of dual-port emulation memory. If a foreground monitor program is selected, the **dp** attribute is automatically assigned to the memory range reserved for the monitor program.

dsi Interlock emulation memory and target system /DSACK.

The **dsi** attribute specifies that accesses in that range of emulation memory be synchronized with the target system. This means the termination of accesses in the range will not occur until the target system provides a /DSACK. If the target system does not generate a /DSACK, the emulator will be unable to break into the monitor and a "CPU in wait state" status will result.

When interlocking is disabled, accesses to emulation memory will be terminated by a /DSACK signal generated by the emulator. Any cycle termination signals generated by the target system during emulation memory accesses, including /BERR, will be ignored.

cs0 Use 68340 chip select 0.

The **cs0** attribute allows you to emulate the 68340's global chip select operation. One memory range, either target or emulation, can be given this attribute. Refer to the "To emulate global chip select operation" task description at the end of this section.

dp_dsi Combination of the **dp** and **dsi** attributes.

dp_cs0 Combination of the **dp** and **cs0** attributes.

dsi_cs0 Combination of the **dsi** and **cs0** attributes.

dp_dsi_cs0 Combination of the **dp**, **dsi**, and **cs0** attributes.

Examples

Consider the following section summary from the linker load map output listing.

```
SECTION SUMMARY
---------------

SECTION    ATTRIBUTE                     START      END        LENGTH     ALIGN

           ABSOLUTE DATA                 00000000   0000002F   00000030   0 (BYTE)
0          NORMAL                        00000030   00000030   00000000   2 (WORD)
env        NORMAL CODE                   00000400   00000FB8   00000BB9   2 (WORD)
prog       NORMAL CODE                   00000FBA   00001A83   00000ACA   2 (WORD)
const      NORMAL ROM                    00001A84   00001AC9   00000046   2 (WORD)
lib        NORMAL CODE                   00001ACA   0000265D   00000B94   2 (WORD)
libc       NORMAL CODE                   0000265E   0000487B   0000221E   2 (WORD)
libm                                     0000487C   0000487C   00000000   0 (BYTE)
mon        NORMAL CODE                   0000487C   000049C5   0000014A   2 (WORD)
envdata    NORMAL DATA                   00007000   00007155   00000156   4 (LONG)
data       NORMAL DATA                   00007156   00007721   000005CC   2 (WORD)
idata                                    00007722   00007722   00000000   0 (BYTE)
udata                                    00007722   00007722   00000000   0 (BYTE)
libdata    NORMAL DATA                   00007724   00007727   00000004   4 (LONG)
libcdata   NORMAL DATA                   00007728   00008153   00000A2C   2 (WORD)
mondata    NORMAL DATA                   00008154   00008177   00000024   2 (WORD)
stack      NORMAL DATA                   0000B000   00012FFF   00008000   4 (LONG)
heap       NORMAL DATA                   00013000   00016FFD   00003FFE   4 (LONG)
```

Notice the ABSOLUTE DATA, CODE, and ROM sections occupy locations 0 through 49C5H. Because the contents of these sections will eventually reside in target system ROM, this area should be characterized as ROM when mapped. This will prevent these locations from being written over accidentally. If breaks on writes to ROM are enabled, instructions that attempt to write to these locations will cause emulator execution to break into the monitor.

Also, notice the DATA sections occupy locations 7000H through 8177H and 0B000H through 16FFDH. Since these sections are written to, they should be characterized as RAM when mapped.

Enter the following commands to map memory for the above program.

delete all <RETURN>
0 ***thru*** 6fffh ***emulation rom*** <RETURN>
7000h ***thru*** 16fffh ***emulation ram*** <RETURN>

The resulting memory mapper screen is shown below.

```
Emulation memory blocks: available = 672  mapped = 368   size = 256  bytes
entry       range         type       function code      attribute
 1       0H-    6FFFH EMUL/ROM
 2    7000H-   16FFFH EMUL/RAM
```

To synchronize emulation memory accesses in the range 0 through 6FFFH, you would enter the following command in place of the command above:

0 ***thru*** 6fffh ***emulation rom dsi*** <RETURN>

To specify that the range 0 through 6FFFH is in supervisor program space, you would enter:

0 ***thru*** 6fffh ***supervisor program emulation rom*** <RETURN>

To exit out of the memory mapper, enter:

end <RETURN>

To characterize unmapped ranges

1 Access the configuration questions.

2 Answer "yes" to the "Modify memory configuration?" question.

3 Use the **default** softkey to characterize unmapped ranges.

The **default** softkey in the memory mapper allows you to characterize unmapped memory ranges. Unmapped memory ranges are treated as target system RAM by default. Unmapped memory ranges cannot be characterized as emulation memory.

Examples

To characterize unmapped ranges as target RAM:

default target ram <RETURN>

To characterize unmapped ranges as guarded memory:

default guarded <RETURN>

To exit out of the memory mapper, enter:

end <RETURN>

To delete memory map ranges

1 Access the configuration questions.

2 Answer "yes" to the "Modify memory configuration?" question.

3 Use the **delete** softkey to delete mapped ranges.

Note that programs should be reloaded after deleting mapper terms. The memory mapper may re-assign blocks of emulation memory after the insertion or deletion of mapper terms.

Examples

To delete term 1 in the memory map:

delete `1 <RETURN>`

To delete all map terms:

delete all `<RETURN>`

To exit out of the memory mapper, enter:

end `<RETURN>`

To map memory ranges that use function codes

- Specify function codes with address ranges when mapping memory.

Memory mapper softkeys that represent the different function codes are:

supervisor
user
program
data
supervisor program
supervisor data
user program
user data

When you specify function codes with mapper ranges, the 68340 function code outputs (FC0, FC1, FC2) are decoded to select particular blocks of memory. Function codes let you overlay address ranges. When you specify function codes as part of the address, the emulator memory mapper knows that overlaid blocks are different memory regions and will define them separately.

If you specify a function code when mapping a range of memory, you must include the function code when referring to locations in that range. If you don't include the function code, an "ambiguous address" error message is displayed.

If you use different function codes, it's possible to map address ranges that overlap. When address ranges with different function codes overlap, you must load a separately linked module for the space associated with each function code. The modules are linked separately because linker errors occur when address ranges overlap.

When address ranges are mapped with different function codes, and there are no overlapping ranges, your program modules may exist in one absolute file. However, you have to use multiple load commands—one for each function code specifier. This is necessary to load the various sections of the absolute file into the appropriate function code qualified memory ranges. When you do this, be sure that all address ranges not mapped (that is, the "other" memory mapper term) are mapped as target RAM. When "other" is mapped as guarded, guarded memory access errors (from the attempt to load the absolute file sections that are outside the specified function code range) can prevent the absolute file sections that are inside the specified function range from being loaded.

Examples

Suppose you're developing a system with the following characteristics:

- Input port at 100 hex.
- Output port at 400 hex.
- Supervisor program from 1000 through 1fff hex.
- User program from 3000 through 3fff hex.
- User data from 3000 through 3fff hex.

Notice that the last two terms have address ranges that overlap. You can use function codes to cause these terms to be mapped to different blocks of memory.

Suppose also that the only things that exist in your target system at this time are the input and output ports and some control logic; no memory is available. You can reflect this by mapping the I/O ports to target system memory space and the rest of memory to emulation memory space with the following mapper commands:

0h ***thru*** 0fffh ***target ram*** <RETURN>
1000h ***thru*** 1fffh ***supervisor program emulation rom*** <RETURN>
3000h ***thru*** 3fffh ***user program emulation ram*** <RETURN>
3000h ***thru*** 3fffh ***user data emulation ram*** <RETURN>

After the configuration is saved, display memory at 1000H by entering the following command (using the command line):

display memory 1000h ***blocked bytes*** <RETURN>

Notice that an "ambiguous address" error occurs because the "sp" function code was not included with the address. The following command should have been entered instead:

display memory fcode sp 1000h ***blocked bytes*** <RETURN>

To emulate global chip select operation

1 Use the **cs0** attribute when mapping the boot ROM address range.

2 Make sure the CF_MBAR register is valid, and modify the CF_CS0ADDR and CF_CS0MASK registers to appropriate values.

Or:

Load a previously saved configuration that has appropriate values of CF_MBAR, CF_CS0ADDR, and CF_CS0MASK.

3 If the you’re emulating boot ROM with emulation memory, load the boot ROM code.

4 Run from reset.

The advantages are:

- You can put the boot ROM contents in emulation memory.
- The base address of the boot ROM does not have to be at address 0 to fetch vectors from reset.
- If boot ROM is already in the target system, you can prevent guarded memory accesses when running from reset.

Limitations:

The maximum amount of emulation memory that can be mapped is half the amount of memory installed in bank 1 or one quarter the amount of memory installed in bank 0, whichever is larger.

Examples

This example shows how to use the **cs0** memory map attribute to emulate the 68340's global chip select operation.

To map the boot ROM address range in emulation memory, access the configuration questions, and answer the questions as shown below.

Modify memory configuration? yes

Entering the following mapper commands. Specify unmapped ranges as guarded memory to show that the fetches of the supervisor stack pointer and program counter values after reset really come from the boot ROM address range.

```
80000h thru 8ffffh emulation rom cs0 <RETURN>
default guarded <RETURN>
```

Configuration file name? boot_cfg

To modify the emulator configuration registers so appropriate information is sent to the analyzer:

```
modify register CF_SIM CF_MBAR to 100001h <RETURN>
modify register CF_SIM CF_CS0ADDR to 80001h <RETURN>
modify register CF_SIM CF_CS0MASK to 0fffdh <RETURN>
```

To load the supervisor stack pointer and program counter values that will be fetched from the boot ROM after reset:

```
modify memory 80000h long to 0 , 81000h <RETURN>
```

To load the boot ROM program into emulation memory (NOP, NOP, BRA.B 81000H):

```
modify memory 81000h words to 4e71h , 4e71h, 60fah
<RETURN>
```

To trace execution after reset:

```
trace <RETURN>
```

To run from reset:

run from reset <RETURN>

To display the trace:

display trace <RETURN>

```
Trace List                 Offset=0
Label:  Address                        Opcode or Status                        time count
Base:     hex                              mnemonic                              relative
after  00000000  ORI.B    #$00,D0                                             ------------
+001   00000002   $0000         supr prgm word rd (ds16)                        720     nS
+002   00000004  ORI.B    #$1000,A0                                             680     nS
+003   00000006   $1000         supr prgm word rd (ds16)                        720     nS
+004   00081000  NOP                                                            840     nS
+005   00081002  NOP                                                            720     nS
+006   00081004  BRA.B    $00081000                                             720     nS
+007   00081006  NOP                                                            720     nS
+008   00081000  NOP                                                            720     nS
+009   00081002  NOP                                                            720     nS
+010   00081004  BRA.B    $00081000                                             720     nS
+011   00081006  NOP                                                            680     nS
+012   00081000  NOP                                                            720     nS
+013   00081002  NOP                                                            720     nS
+014   00081004  BRA.B    $00081000                                             720     nS
+015   00081006  NOP                                                            720     nS
```

Notice the supervisor stack pointer and program counter values that were loaded at 80000H and 80004H are fetched from memory locations 0 through 7 and the program begins running at 81000H.

Suppose your boot ROM is at a higher address:

To map the boot ROM address range in emulation memory, access the configuration questions, and answer the questions as shown below.

Modify memory configuration? yes

Entering the following mapper commands. Specify unmapped ranges as guarded memory to show that the fetches of the supervisor stack pointer and program counter values after reset really come from the boot ROM address range.

0ff000000h ***thru*** 0ff00ffffh ***emulation rom cs0*** <RETURN>
default guarded <RETURN>

Configuration file name? boot_cfg

To modify the emulator configuration registers so appropriate information is sent to the analyzer:

modify register CF_SIM CF_MBAR to 100001h <RETURN>
modify register CF_SIM CF_CS0ADDR to 0ff000001h <RETURN>
modify register CF_SIM CF_CS0MASK to 0fffdh <RETURN>

To load the supervisor stack pointer and program counter values that will be fetched from the boot ROM after reset:

modify memory 0ff000000h ***long to*** 0 , 0ff001000h <RETURN>

To load the boot ROM program into emulation memory (NOP, NOP, BRA.B 0FF001000H):

modify memory 0ff001000h ***words to*** 4e71h , 4e71h, 60fah <RETURN>

To trace execution after reset:

trace <RETURN>

To run from reset:

run from reset <RETURN>

To display the trace:

display trace <RETURN>

```
Trace List                 Offset=0
Label:   Address                         Opcode or Status                        time count
Base:      hex                              mnemonic                              relative
after  FF000000  ORI.B    #$00,D0                                              ------------
+001   FF000002   $0000          supr prgm word rd (ds16)                        720      nS
+002   FF000004  Illegal Instruction: $FF001000                                  720      nS
+003   FF000006   $1000          supr prgm word rd (ds16)                        680      nS
+004   FF001000  NOP                                                             840      nS
+005   FF001002  NOP                                                             720      nS
+006   FF001004  BRA.B    $FF001000                                              720      nS
+007   FF001006  NOP                                                             720      nS
+008   FF001000  NOP                                                             720      nS
+009   FF001002  NOP                                                             720      nS
+010   FF001004  BRA.B    $FF001000                                              720      nS
+011   FF001006  NOP                                                             720      nS
+012   FF001000  NOP                                                             680      nS
+013   FF001002  NOP                                                             720      nS
+014   FF001004  BRA.B    $FF001000                                              720      nS
+015   FF001006  NOP                                                             720      nS
```

Notice the supervisor stack pointer and program counter values that were loaded at 0FF000000H appear to be fetched from memory locations 0FF000000H through 0FF000007H when they are really fetched from locations 0 through 7. This is because the upper 8 bits of the CF_CS0ADDR register are sent to the analyzer instead of A31-A24 (this is true even if Port A is set up to be address lines).

Configuring the Emulator Pod

In order to configure the emulator pod, you must first start the configuration interface and access the "Emulator Pod Settings" configuration section (refer to the previous "Using the Configuration Interface" section).

This section shows you how to:

- Set the reset values of the Supervisor Stack Pointer and the Program Counter.
- Set the user memory access size.

To set the reset values of the SSP and PC

1 Enter an even address in response to the "Reset value for Supervisor Stack Pointer?" question.

2 Enter an even address in response to the "Reset value of the Program Counter?" question.

When using a foreground monitor, the supervisor stack pointer must be set to an address in emulation or target system RAM in order for the emulator to transition into the run state, to step, or to perform other functions after emulation reset.

The "Reset value ... ?" configuration questions set the initial SSP and PC values after emulation reset. Upon the transition from emulation reset into the emulation monitor, the supervisor stack pointer register and the program counter are set to the values specified, which must be 32-bit hexadecimal even addresses.

If a run from reset command is given, this configuration item has no affect and the initial supervisor stack pointer and program counter will be retrieved from reset vector in the vector table.

When using the background monitor, the initial values set in response to the "Reset value ... ?" configuration questions are also used to set up the SSP and PC after emulation reset; however, this is not necessary for proper emulator operation. If a target system reset occurs while running in the background monitor, the supervisor stack pointer and program counter are unaffected.

To specify the user memory access size

- Answer the "User memory access size?" question.

When accessing target system memory locations or single-port emulation memory locations, the access mode specifies the type of microprocessor cycles that are used to read or write the value(s). For example, when the access mode is byte and a target system location is modified to contain the value 12345678H, byte instructions are used to write the byte values 12H, 34H, 56H, and 78H to target system memory.

Answer "bytes" if the emulator should make 8-bit accesses to target system memory.

Answer "words" if the emulator should make 16-bit accesses to target system memory.

Answer "longs" if the emulator should make 32-bit accesses to target system memory.

Setting the Debug/Trace Options

In order to set the debug/trace options, you must first start the configuration interface and access the "Debug/Trace Options" configuration section (refer to the previous "Using the Configuration Interface" section).

This section shows you how to:

- Enable/disable breaks on writes to ROM.
- Trace background cycles.

To enable/disable breaks on writes to ROM

1. Access the configuration questions.

2. Answer "yes" to the "Modify trace/debug options?" question.

3. Answer "yes" to the "Break processor on write to ROM?" question to enable breaks; answer "no" to disable breaks.

   When breaks on writes to ROM are enabled:

   > The emulator will break into the emulation monitor whenever the user program attempts to write to a memory region mapped as ROM.
   >
   > Even though execution breaks into the monitor, the memory location is modified if it's in emulation ROM or target system RAM mapped as ROM.

   When breaks on writes to ROM are disabled:

   > The emulator will not break to the monitor upon a write to ROM.

To trace background cycles

- Answer "background" or "both" to the "Trace background or foreground operation?" question.

Answering "background" specifies that the analyzer trace only background cycles. This is rarely a useful setting for user program debugging.

Because the background monitor is implemented using the 68340 background debug mode (BDM), only memory accesses are captured when tracing background cycles.

Answering "both" specifies that the analyzer trace both foreground and background cycles. You may wish to specify this option so that all emulation processor cycles may be viewed in the trace display.

6

Using the Emulator

Using the Emulator

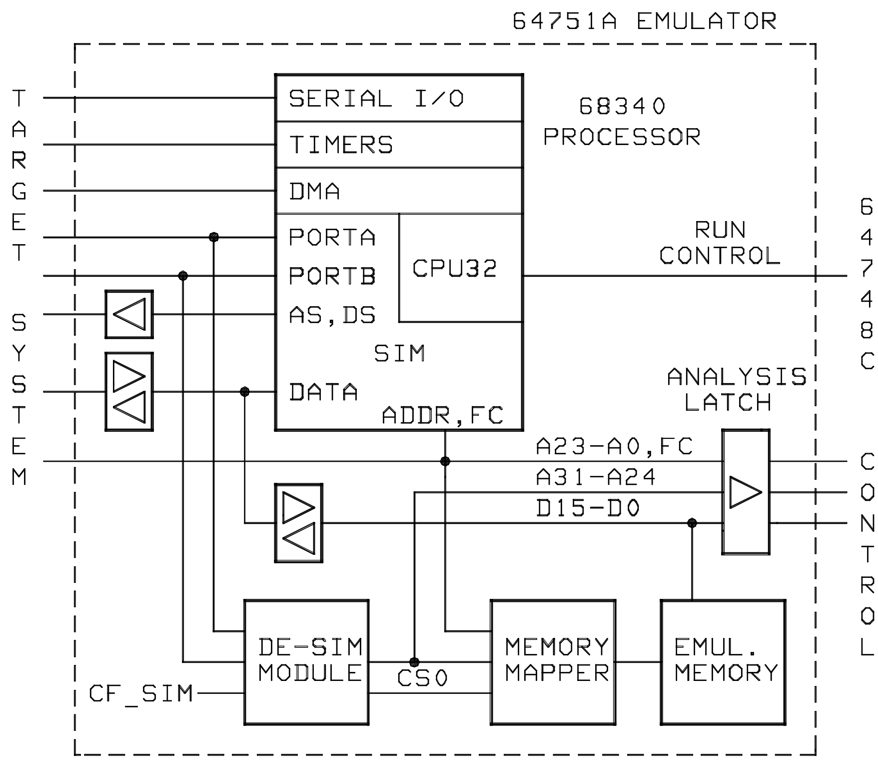
This chapter describes general tasks you may wish to perform while using the emulator. These tasks are grouped into the following sections:

- Using the emulator configuration registers.
- Loading absolute files.
- Using symbols.
- Using context commands.
- Executing user programs (starting, stopping, stepping, and resetting the emulator).
- Using software breakpoints.
- Displaying and modifying registers.
- Displaying and modifying memory.
- Displaying data values.
- Changing the interface settings.
- Using system commands.
- Using simulated I/O.
- Using Basis Branch Analysis.

Using the Emulator Configuration Registers

The 68340 processor contains a System Integration Module (SIM) which has the external bus interface, four chip selects, input/output ports, and other circuitry to reduce external logic in a typical microprocessor system. The SIM can be programmed or configured in a variety of ways to suit the need of various systems.

The HP 64751A emulator contains circuitry that accommodates the flexibility of the 68340 SIM and maintains consistent emulation features.



In the previous figure, there is a block labeled DE-SIM module. It receives as inputs from the 68340:

- Port A which can be address lines 31-24, interrupt acknowledge inputs, or general purpose I/O lines.
- Port B which can be interrupt requests, chip selects, or general purpose I/O lines.

The DE-SIM module provides as outputs:

- Address lines A31-A24 to the memory mapper and the analyzer.
- A qualified chip select 0 (CS0) to the memory mapper.

The 68340 SIM is configured through the registers in the SIM register class; these registers control how the 68340 uses external signal lines to access memory.

The emulator's DE-SIM module is configured through the registers in the CF_SIM register class. The DE-SIM module controls how the emulator interprets the signals from the 68340 when accessing emulation memory and passing information to the analysis trace.

Normally, the SIM and CF_SIM registers should be programmed with the same values so they will be working together.

One of the primary functions of the DE-SIM is to provide A31-A24 to the memory mapper and analyzer so they will have the complete 32-bit address bus. This is easy if Port A of the 68340 is programmed as address lines; however, if it's programmed as an input port, for example, the upper address lines are not available external to the 68340 (this is the case following reset). The four chip selects, however, have access to the full 32 bit address inside the 68340. You can therefore locate memory using a chip select at an address that is not possible to decode externally. If properly programmed, the DE-SIM can use information in the programming of the chip selects to re-create the upper address lines. This provides the ability to map emulation memory at these addresses and also provides a correct address in the analysis trace so that symbolic debugging is possible.

Normally, the DE-SIM would be programmed through the CF_SIM registers to match the programming of the 68340 SIM as it will exist after all of the boot-up configuration is complete. This can be done before the boot-up code is run. In fact, the programming of the CF_SIM registers is part of the configuration and will be loaded along with the memory map and other configuration items when a configuration file is loaded.

The default programming of the DE-SIM matches the reset values of the 68340 SIM (refer to the Motorola *MC68340 Integrated Processor User's Manual* for specific values).

If desired, the programming of the DE-SIM can be transferred into the 68340 SIM with the **sync_sim_registers to_68340_from_config** command. This happens automatically each time a break to the monitor from emulation reset occurs. This ensures that the 68340 is prepared to properly access memory when a program is downloaded to the emulator.

Alternatively, the emulator's DE-SIM can be programmed from the 68340 SIM with the **sync_sim_registers from_68340_to_config** command. This is useful if initialization code that configures the 68340 SIM exists, but you don't what its values are. In this case, you can use the default configuration, run from reset to execute the initialization code, and use the **sync_sim_registers from_68340_to_config** command to configure the emulator to match the 68340 SIM.

At any time, you can verify if the SIM and DE-SIM are programmed the same with the **sync_sim_registers difference** command. Any differences between the two register sets will be listed.

It should be noted that the DE-SIM module is programmed solely from the CF_SIM register set and is therefore static with respect to the application program. No attempt is made to update the programming of the DE-SIM by tracking instructions that will program the 68340 SIM.

This section shows you how to:

- View the SIM register differences.
- Synchronize to the 68340 SIM registers.
- Synchronize to the emulator configuration registers.

To view the SIM register differences

- Enter the **sync_sim_registers difference** command.

Before displaying the SIM configuration register differences, make sure the contents of the MBAR register is valid (in other words, its least significant bit should be 1).

Examples

To display the SIM register differences:

```
modify register SIM MBAR to 40001h <RETURN>
sync_sim_registers difference <RETURN>
```

```
MBAR    =    40001  CF_MBAR    =   100001
CS0MASK = ffffffff  CF_CS0MASK = ffff000d
CS0ADDR = 18515e60  CF_CS0ADDR = ff000000
CS1MASK = fffffffb  CF_CS1MASK = 00000000
CS1ADDR = 8830fcc8  CF_CS1ADDR = 00000000
CS2MASK = fff7ffff  CF_CS2MASK = 00000000
CS2ADDR = e41cff24  CF_CS2ADDR = 00000000
CS3MASK = fffffffd  CF_CS3MASK = 00000000
CS3ADDR = 54005ca0  CF_CS3ADDR = 00000000
```

To synchronize to the 68340 SIM registers

- Enter the **sync_sim_registers from_68340_to_config** command.

The contents of the 68340 SIM registers are copied to the emulator’s configuration registers. The contents of the MBAR register must be valid (that is, its least significant bit should be 1).

To synchronize to the emulator configuration registers

- Enter the **sync_sim_registers to_68340_from_config** command.

The contents of the emulator's configuration registers are copied to the 68340 SIM registers. The contents of the CF_MBAR register must be valid (that is, its least significant bit should be 1).

Loading and Storing Absolute Files

This section describes the tasks related to loading absolute files into the emulator and storing memory contents into absolute files. This section shows you how to:

- Load absolute files into memory.
- Load absolute files without symbols.
- Store memory contents into absolute files.

To load absolute files

- Choose **File→Load→Executable** and use the dialog box to select the absolute file.
- Using the command line, enter the **load <absolute_file>** command.

You can load absolute files into emulation or target system memory. You can load IEEE-695 format absolute files. You can also load HP format absolute files. The **store memory** command creates HP format absolute files.

If you wish to load only that portion of the absolute file that resides in memory mapped as emulation RAM or ROM, use the command line's **load emul_mem** syntax.

If you wish to load only the portion of the absolute file that resides in memory mapped as target RAM, use the command line's **load user_mem** syntax.

If you want both emulation and target memory to be loaded, do not specify **emul_mem** or **user_mem**.

Examples

To load the demo program absolute file, enter the following command:

```
load ecs.x <RETURN>
```

To load only portions of the absolute file that reside in target system RAM:

```
load user_mem absfile <RETURN>
```

To load only portions of the absolute file that reside in emulation memory:

```
load emul_mem absfile <RETURN>
```

To load absolute files without symbols

- Choose **File→Load→Program Only** and use the dialog box to select the absolute file.

- Using the command line, enter the **load <absolute_file> nosymbols** command.

To store memory contents into absolute files

- Using the command line, enter the **store memory** command.

You can store emulation or target system memory contents into HP format absolute files on the host computer. Absolute files are stored in the current directory. If no extension is given for the absolute file name, it is given a ".X" extension.

Storing memory contents into absolute files is useful when copying ROM contents to RAM so that software breakpoints can be used. In other words, you store ROM contents to an absolute file, re-map memory to substitute emulation RAM for the target ROM, and load the absolute file.

Examples

To store the contents of memory locations 900H through 9FFH to an absolute file on the host computer named "absfile":

store memory `900h` ***thru*** `9ffh` ***to*** `absfile <RETURN>`

After the command above, a file named "absfile.X" exists in the current directory on the host computer.

Using Symbols

If symbol information is present in the absolute file, it is loaded along with the absolute file (unless you use the **nosymbols** option). Both global symbols and symbols that are local to a program module can be displayed.

Long symbol names can be truncated in the symbols display; however, you can increase the width of the symbols display by starting the interface with more columns (refer to the "Setting X Resources" chapter).

This section describes how to:

- Load symbols.
- Display global symbols.
- Display local symbols.
- Display a symbol's parent symbol.
- Copy-and-paste a full symbol name to the entry buffer.

To load symbols

- Choose **File→Load→Symbols Only** and use the dialog box to select the absolute file.

- Using the command line, enter the **load symbols <absolute_file>** command.

Unless you use the **nosymbols** option when loading absolute files, symbols are loaded automatically. However, if you did use the **nosymbols** option when loading the absolute file, you can load the symbols without loading the absolute file again.

This option is particularly useful for loading symbols for files located in target ROM so that you can use symbols with that code.

Examples

To load symbols from the demo program:

load symbols `ecs.x` <RETURN>

To display global symbols

- Choose **Display→Global Symbols**.
- Using the command line, enter the **display global_symbols** command.

Listed are: address ranges associated with a symbol, the segment the symbol is associated with, and the offset of that symbol within the segment.

If there is more than a screen full of information, you can use the up arrow, down arrow, <NEXT>, or <PREV> keys to scroll the information up or down on the display.

Examples

To display global symbols in the demo program:

display global_symbols <RETURN>

```
Global symbols in ecs.x
Procedure symbols
Procedure name _________________ Address range __ Segment _____________ Offset
__fflush                         0000440C - 000044A5 libc                  0000
_bufsync                         00002CDC - 00002D09 libc                  0000
_dbl_to_str                      00002EEE - 00003393 libc                  01E4
_doprnt                          00003640 - 00004371 libc                  0036
_exec_funcs                      0000268A - 000026A9 libc                  002C
_findbuf                         000044A6 - 00004539 libc                  0000
_startup                         000005FC - 0000071B env                   0000
_swrite                          0000471C - 00004751 libc                  0000
_wrtchk                          00004752 - 000047E9 libc                  0000
_xflsbuf                         000047EA - 0000487B libc                  0000
atexit                           0000265E - 00002689 libc                  0000
calloc                           00002BC6 - 00002BF7 libc                  0412
clear_screen                     000009FE - 00000A33 env                   01CA
close                            000008CE - 00000903 env                   009A
combsort                         00001244 - 0000143B prog                  028A
do_sort                          00001442 - 000014E1 prog                  0488
```

To display local symbols

- When displaying symbols, position the mouse pointer over a symbol on the symbol display screen and click the *select* mouse button.

- When displaying symbols, position the mouse pointer over the symbol, press and hold the *select* mouse button, and choose **Display Local Symbols** from the popup menu.

- Position the mouse cursor in the entry buffer and enter the module whose local symbols are to be displayed; then, choose **Display→Local Symbols ()**.

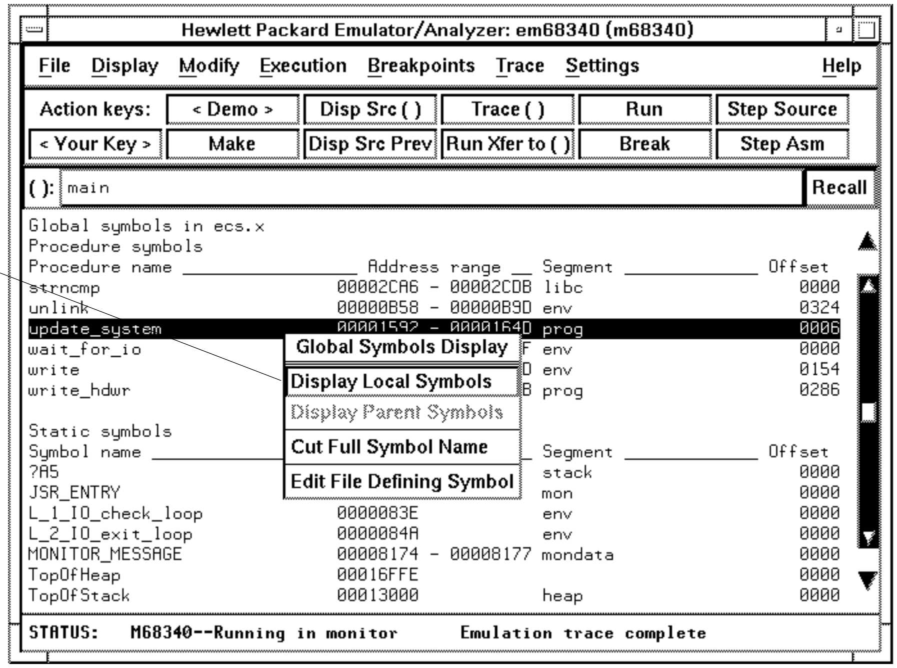
- Using the command line, enter the **display local_symbols_in <module>** command.

  To display the address ranges associated with the high-level program's source file line numbers, you must display the local symbols in the file.

Examples

To use the Symbols Display popup menu:

View the local symbols associated with the highlighted symbol by choosing this menu item.



Using the command line:

To display local symbols in a module:

display local_symbols_in `update_sys <RETURN>`

```
Symbols in update_sys(module)
Procedure symbols
Procedure name _________________ Address range __ Segment _____________ Offset
get_targets                    00001654 - 000016DD prog                  00C8
graph_data                     0000197E - 00001A37 prog                  03F2
read_conditions                000016E4 - 00001773 prog                  0158
save_points                    00001892 - 00001977 prog                  0306
set_outputs                    0000177A - 0000180B prog                  01EE
update_system                  00001592 - 0000164D prog                  0006
write_hdwr                     00001812 - 0000188B prog                  0286

Filename symbols
Filename _____________________________________________________________________
update_sys.c
```

To display local symbols in a procedure:

display local_symbols_in `update_sys.save_points <RETURN>`

```
Symbols in update_sys(module).save_points(procedure)
Procedure special symbols
Procedure special name _________ Address range __ Segment _____________ Offset
ENTRY                          00001892            prog                  0306
EXIT                           00001976            prog                  03EA
TEXTRANGE                      00001892 - 00001977 prog                  0306
```

To display address ranges associated with the high-level source line numbers:

display local_symbols_in `update_sys."update_sys.c": <RETURN>`

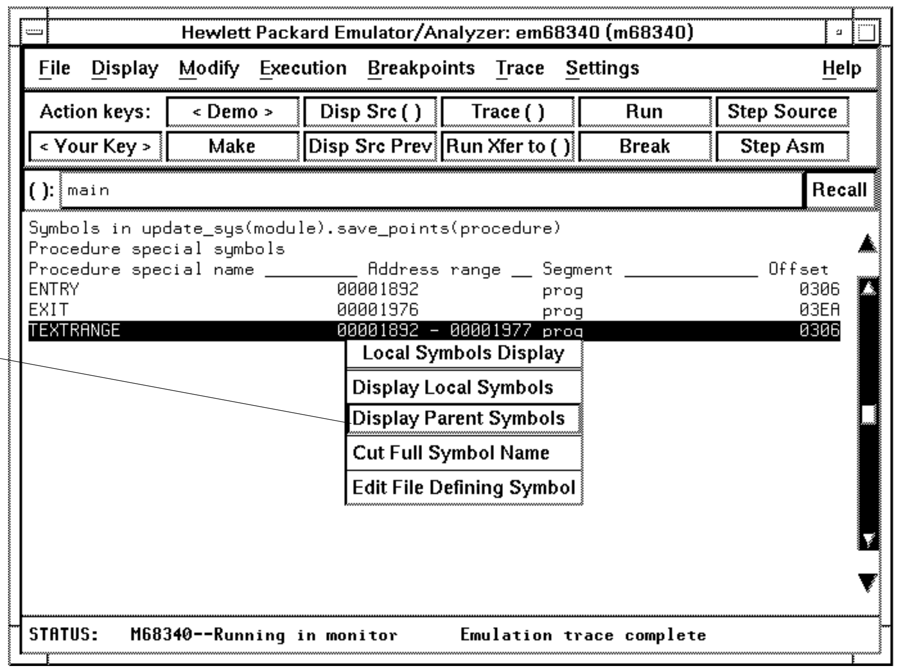
```
Symbols in update_sys(module)."update_sys.c":
Source reference symbols
Line range ______________________ Address range __ Segment _____________ Offset
#1-#47                        00001592 - 000015B1 prog                   0006
#48-#53                       000015B2 - 000015C5 prog                   0026
#54-#56                       000015C6 - 000015D1 prog                   003A
#57-#59                       000015D2 - 000015ED prog                   0046
#60-#60                       000015EE - 000015F3 prog                   0062
#61-#61                       000015F4 - 00001603 prog                   0068
#62-#63                       00001604 - 00001609 prog                   0078
#64-#64                       0000160A - 00001617 prog                   007E
#65-#68                       00001618 - 00001629 prog                   008C
#69-#72                       0000162A - 0000163D prog                   009E
#73-#75                       0000163E - 00001643 prog                   00B2
#76-#77                       00001644            prog                   00B8
#78-#94                       00001654 - 0000166F prog                   00C8
#95-#95                       00001670 - 00001687 prog                   00E4
#96-#99                       00001688 - 0000168D prog                   00FC
#100-#100                     0000168E - 0000168F prog                   0102
```

To display a symbol's parent symbol

- When displaying symbols, position the mouse pointer over the symbol, press and hold the *select* mouse button, and choose **Display Parent Symbols** from the popup menu.

Examples

View the parent symbol associated with the highlighted symbol by choosing this menu item.



To copy-and-paste a full symbol name to the entry buffer

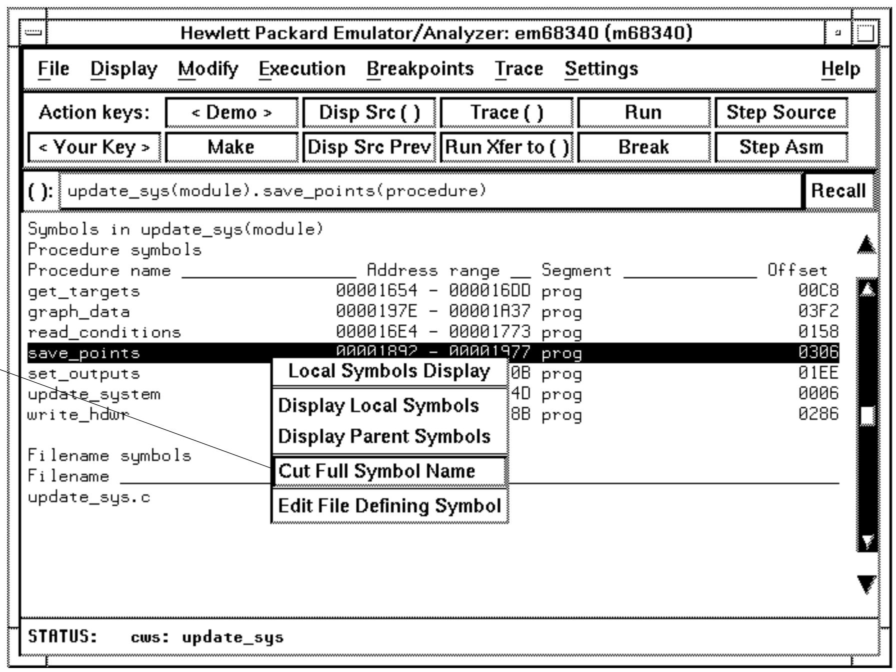
- When displaying symbols, position the mouse pointer over the symbol, press and hold the *select* mouse button, and choose **Cut Full Symbol Name** from the popup menu.

Once the full symbol name is in the entry buffer, you can use it with pulldown menu items or paste it to the command line area.

By cutting the full symbol name, you get the complete names of symbols that have been truncated. Also, you are guaranteed of specifying the proper scope of the symbol.

Examples

Copy the full name of the highlighted symbol to the entry buffer by choosing this menu item.



Using Context Commands

The commands in this section display and control the directory and symbol contexts for the interface.

Directory context. The current directory context is the directory accessed by all system references for files—primarily load, store, and copy commands—if no explicit directory is mentioned. Unless you have changed directories since beginning the emulation session, the current directory context is that of the directory from which you started the interface.

Symbol context. The emulator/analyzer interface and the Symbol Retrieval Utilities (SRU) together support a current working symbol context. The current working symbol represents an enclosing scope for local symbols. If symbols have not been loaded into the interface, you cannot display or change the symbol context.

This section shows you how to:

- Display the current directory and symbol context.
- Change the directory context.
- Change the symbol context.

To display the current directory and symbol context

- Choose **Display→Context**.

- Using the command line, enter the **pwd** and **pws** commands.

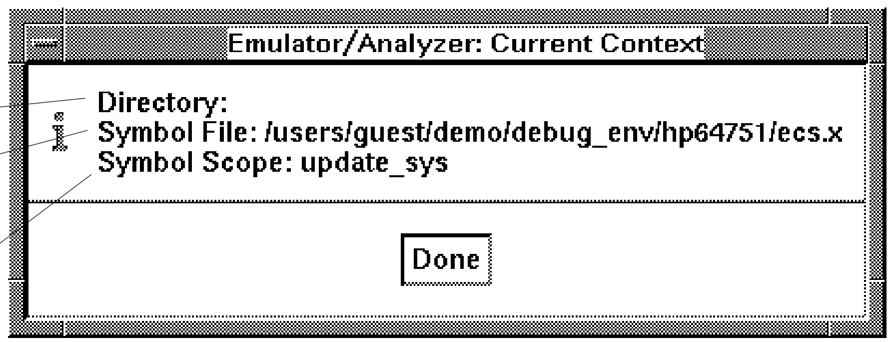
The current directory and working symbol contexts are displayed, and also the name of the last executable file from which symbols were loaded.

Example

Directory context.

Executable from which symbols were last loaded.

Symbol context.



To change the directory context

- Choose **File→Context→Directory** and use the dialog box to select a new directory.

- Using the command line, enter the **cd <directory>** command.

The Directory Selection dialog box contains a list of directories accessed during the emulation session as well as any predefined directories present at interface startup.

You can predefine directories and set the maximum number of entries for the Directory Selection dialog box by setting X resources (see the "Setting X Resources" chapter).

To change the current working symbol context

- Choose **File→Context→Symbols** and use the dialog box to select the new working symbol context.

- Using the command line, enter the **cws <symbol_context>** command. (Because **cws** is a hidden command and doesn't appear on a softkey label, you have to type it in.)

  You can predefine symbol contexts and set the maximum number of entries for the Symbol Scope Selection dialog box by setting X resources (see the "Setting X Resources" chapter).

  Displaying local symbols or displaying memory in mnemonic format causes the working symbol context to change as well. The new context will be that of the local symbols or memory locations displayed.

Executing User Programs

You can use the emulator to run programs, break program execution into the monitor, step through the program by high-level source lines or by assembly language instructions, and reset the emulation processor.

When displaying memory in mnemonic format, a highlighted bar shows the current program counter address. When you step, the mnemonic memory display is updated to highlight the new program counter address.

When displaying resisters, the register display is updated to show you the contents of the registers after each step.

You can open multiple interface windows to display memory in mnemonic format and registers at the same time. Both windows are updated after stepping.

This section describes how to:

- Start the emulator running the user program.
- Stop (break from) user program execution.
- Step through user programs.
- Reset the emulation processor.

To run programs from the current PC

- Choose **Execution→Run→from PC**.
- Using the command line, enter the **run** command.

When the emulator is executing the user program, the message "Running user program" is displayed on the status line.

To run programs from an address

- Position the mouse pointer in the entry buffer and enter the address you want to run from; then, choose **Execution→Run→from ()**.

- Using the command line, enter the **run from <address>** command.

Examples

To run from address 9C0H:

run from `9c0h <RETURN>`

To run programs from the transfer address

- Choose **Execution→Run→from Transfer Address**.

- Using the command line, enter the **run from transfer_address** command.

Most software development tools allow you to specify a starting or entry address for program execution. That address is included with the absolute file's symbolic information and is known by the interface as the *transfer address*.

To run programs from reset

- Choose **Execution→Run→from Reset**.

- Using the command line, enter the **run from reset** command.

The run from reset command resets the emulation processor and lets the emulator run and fetch its stack pointer and program counter value from memory.

A **reset** command followed by a **run** command will load the interrupt stack pointer and program counter values specified during configuration into the emulation processor and run from the loaded program counter value. This is true for both background and foreground monitors.

To run programs until an address

- Position the mouse pointer in the entry buffer and enter the address you want to run from; then, choose **Execution→Run→until ()**.
- Using the command line, enter the **run until <address>** command.

When you run until an address, a software breakpoint is set at the address and the program is run from the current program counter.

When using the command line, you can combine the various types of run commands; for example, you can run from the transfer address until another address.

Examples

To run from the transfer address until the address of the global symbol main:

```
run from transfer_address until address main <RETURN>
```

To stop (break from) user program execution

- Choose **Execution→Break**.
- Using the command line, enter the **break** command.

This command generates a break to the background monitor.

If the user program executes a STOP or LPSTOP instruction, you cannot break to the emulator's monitor state while the processor is in the stopped state. The break command uses the emulation processor background debug mode (BDM), and the processor must be executing instructions in order to enter the BDM. An interrupt from the target system will cause the 68340 to exit the stopped state; then, the break command will work normally.

Software breakpoints and the **run until** command allow you to stop execution at particular points in the user program.

Examples

To break emulator execution from the user program to the monitor:

break `<RETURN>`

To step high-level source lines

- Choose **Execution→Step Source** and select one of the items from the cascade menu.
- Using the command line, enter the **step source** command.

When stepping through instructions associated with source lines, execution can remain in a loop and the message "Stepping source line 1; Next PC: <address>" is displayed on the status line. In this situation you can abort the step command by pressing <CTRL>c.

Examples

To step through instructions associated with the high-level source lines at the current program counter:

step source `<RETURN>`

To step through instructions associated with high-level source lines at address "main":

step source from `main <RETURN>`

To step assembly-level instructions

- Choose **Execution→Step Instruction** and select one of the items from the cascade menu.

- Using the command line, enter the **step** command.

The step command allows you to step through program execution an instruction or a number of instructions at a time. Also, you can step from the current program counter or from a specific address.

Examples

To step one instruction from the current program counter:

step `<RETURN>`

To step a number of instructions from the current program counter:

step `8 <RETURN>`

To step a number of instructions from a specified address:

step `16` ***from*** `920h <RETURN>`

To reset the emulation processor

- Choose **Execution→Reset**.

- Using the command line, enter the **reset** command.

The **reset** command causes the processor to be held in a reset state until a **break**, **run**, or **step** command is entered. A CMB execute signal will also cause the emulator to run if reset. Also, a request to access memory or registers while reset will cause a break into the monitor.

Using Software Breakpoints

Software breakpoints provide a way to accurately stop the execution of your program at selected locations.

Note

Version A.04.00 or greater of the HP 64700 system firmware provides support for permanent as well as temporary breakpoints. If your version of HP 64700 system firmware is less than A.04.00, only temporary breakpoints are supported.

When you set a software breakpoint at an address, the instruction at that address is replaced with a BGND instruction. When the BGND instruction is executed, control is passed to the emulator's monitor program, and the original instruction is restored in the user program.

If the BGND instruction was not inserted as the result of a **modify software_breakpoints set** command, the "Undefined software breakpoint" message is displayed on the status line.

In order to successfully set a software breakpoint, the emulator must be able to write to the memory location specified. Therefore, software breakpoints cannot be set in target memory while the emulator is reset, and they can never be set in target ROM. (You can, however, copy target ROM to emulation memory by storing the contents of target ROM to an absolute file, re-mapping the range as emulation RAM, and loading the absolute file.)

Another way to break user program execution at a certain point is to break on the analyzer trigger.

This section shows you how to:

- Display the breakpoints list.
- Enable/disable breakpoints.
- Set a permanent breakpoint.
- Set a temporary breakpoint.
- Set all breakpoints.
- Deactivate a breakpoint.

- Re-activate a breakpoint.
- Clear a breakpoint.
- Clear all breakpoints.

CAUTION Software breakpoints should not be set, cleared, enabled, or disabled while the emulator is running user code. If any of these commands are entered while the emulator is running user code, and the emulator is executing code in the area where the breakpoint is being modified, program execution may be unreliable.

To display the breakpoints list

- Choose **Display→Breakpoints** or **Breakpoints→Display**.
- Using the command line, enter the **display software_breakpoints** command.

The breakpoints display shows the address and status of each breakpoint currently defined. If symbolic addresses are turned on (when setting the display modes), the symbolic label associated with a breakpoint is also displayed. Also, the breakpoints display shows whether the breakpoint feature is enabled or disabled.

```
Software breakpoints  :enabled
  address          label                                          status
 00000FC0         main(module)."main.c":              line   96  temporary
 00000FCA @sp     main(module)."main.c":              line   98  pending
 00000FD2         main(module)."main.c":              line  102  permanent
 00000FFC         main(module)."main.c":              line  107  inactivated
```

The status of a breakpoint can be:

temporary — Which means the temporary breakpoint has been set but not encountered during program execution. These breakpoints are removed when the breakpoint is encountered.

pending	Which means the temporary breakpoint has been set but not encountered during program execution. These breakpoints are inactivated when the breakpoint is encountered.
permanent	Which means the permanent breakpoint is active.
inactivated	Which means the breakpoint has been inactivated somehow. Temporary breakpoints are inactivated when they are encountered during program execution. Both temporary and permanent breakpoints may be inactivated using the breakpoints display popup menu.

In the breakpoints display, a popup menu is available. You can set, inactivate, or clear breakpoints as well as enable or disable the breakpoints feature from the popup menu.

To enable/disable breakpoints

- Choose the **Breakpoints→Enable** toggle.

- When displaying the breakpoint list, press and hold the *select* mouse button and then choose **Enable/Disable Software Breakpoints** from the popup menu.

- Using the command line, enter the **modify software_breakpoints enable** or **modify software_breakpoints disable** command.

The breakpoints feature must be enabled before you can set, inactivate, or clear breakpoints.

If breakpoints were set when the feature was disabled, they are "inactivated" when the feature is re-enabled, and you must set them again.

The emulator/analyzer interface will enable software breakpoints whenever the **XEnv_68k_except** symbol is present in the symbol data base.

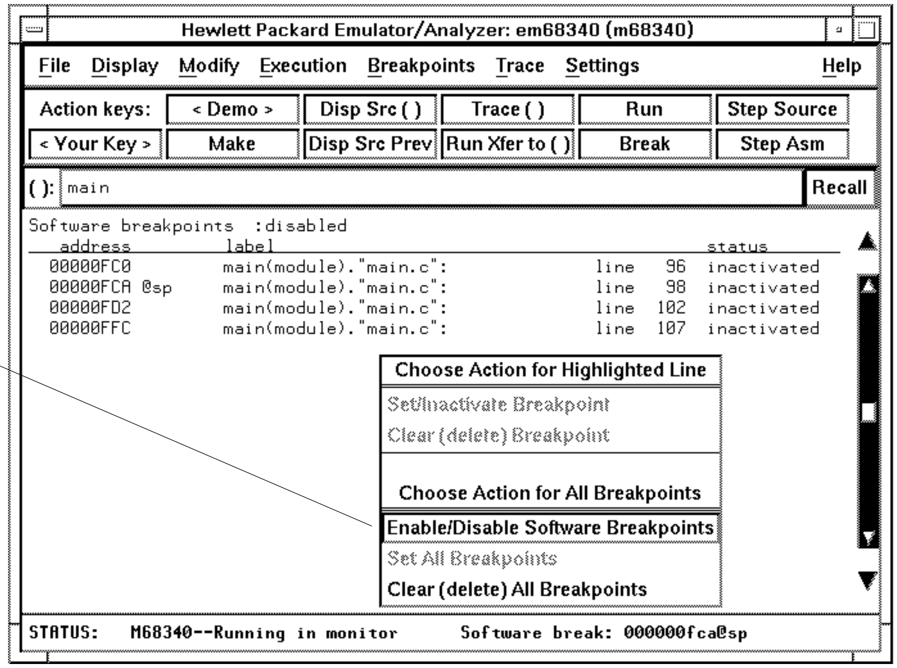
The run-time library provided with the 68332 C Cross Compiler uses software breakpoints to interrupt program execution when exceptions (for example, divide

by zero) are encountered. If software breakpoints are disabled, exception processing may result in "access to guarded memory" errors and/or other unpredictable behavior. To prevent this, a special global symbol, **XEnv_68k_except**, is included in the library.

When the **XEnv_68k_except** symbol is present, the 68340 emulator writes a value to this location. The value tells the run-time library to use the BGND instruction to perform a software break.

Examples

To enable software breakpoints using the breakpoints display popup menu:



Bring up menu and choose this item to change states.

To set a permanent breakpoint

- When displaying memory in mnemonic format, position the mouse pointer over the program line at which you wish to set the breakpoint and click the *select* mouse button. Or, press and hold the *select* mouse button and choose **Set/Clear Software Breakpoint** from the popup menu.

- Place an absolute or symbolic address in the entry buffer; then, choose **Breakpoints→Permanent ()**

- Using the command line, enter the **modify software_breakpoints set <address> permanent** command.

Permanent breakpoints are available if your version of HP 64700 system firmware is A.04.00 or greater.

The breakpoints feature must be enabled before individual breakpoints can be set.

Note that you must only set software breakpoints at memory locations which contain instruction opcodes (not operands or data).

When displaying memory in mnemonic format, asterisks (*) appear next to breakpoint addresses. An asterisk shows the breakpoint is active. Also, if assembly level code is being displayed, the disassembled instruction mnemonic at the breakpoint address will show the breakpoint instruction.

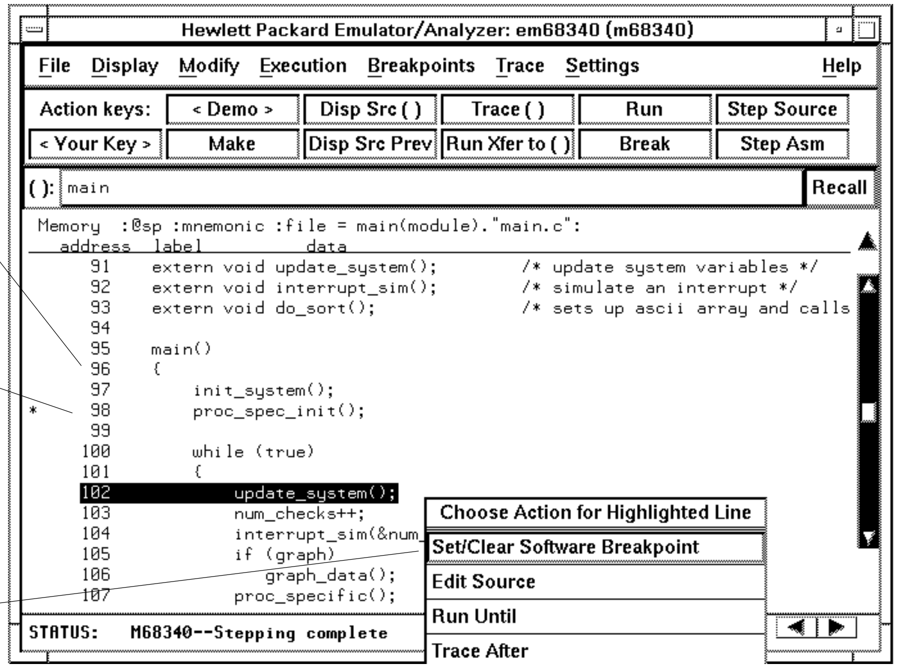
Examples

To set permanent breakpoints using the mnemonic memory display popup menu:

Click this line to set a breakpoint.

Click this line to clear a breakpoint. (Asterisks mark set breakpoints.)

Bring up menu and choose this item to set (or clear) a breakpoint on the highlighted line.



To set a temporary breakpoint

- Place an absolute or symbolic address in the entry buffer; then, choose **Breakpoints→Temporary ()** (or **Breakpoints→Set ()** if your version of HP 64700 system firmware is less than A.04.00).

- Using the command line, enter the **modify software_breakpoints set <address> temporary** or **modify software_breakpoints set <address>** command.

The breakpoints feature must be enabled before individual breakpoints can be set.

Note that you must only set software breakpoints at memory locations which contain instruction opcodes (not operands or data).

When displaying memory in mnemonic format, asterisks (*) appear next to breakpoint addresses. An asterisk shows the breakpoint is active. Also, if assembly level code is being displayed, the disassembled instruction mnemonic at the breakpoint address will show the breakpoint instruction.

To set all breakpoints

- When displaying the breakpoint list, position the mouse pointer within the breakpoints display screen, press and hold the *select* mouse button, and choose **Set All Breakpoints** from the popup menu.

- Choose **Breakpoints**→**Set All**.

- Using the command line, enter the **modify software_breakpoints set** command.

Breakpoints must be enabled before being set.

To deactivate a breakpoint

- When displaying breakpoints, position the mouse pointer over the line displaying the active breakpoint and click the *select* mouse button. Or, press and hold the *select* mouse button and choose **Set/Inactivate Breakpoint** from the popup menu.

A deactivated breakpoint remains in the breakpoint list and can be re-activated later. Deactivating a breakpoint is different than clearing a breakpoint because a cleared breakpoint is removed from the breakpoints list.

To re-activate a breakpoint

- When displaying breakpoints, position the mouse pointer over the line displaying the inactivated breakpoint and click the *select* mouse button. Or, press and hold the *select* mouse button and choose **Set/Inactivate Breakpoint** from the popup menu.

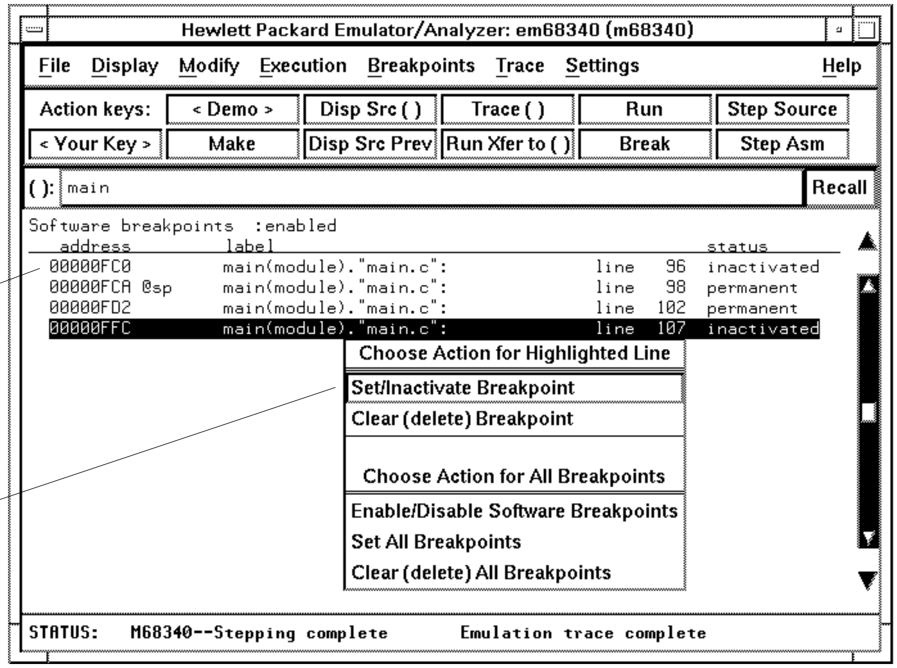
  The "inactivated" breakpoint either becomes "temporary" (or "pending") if it was set as a temporary breakpoint or "permanent" if it was set as a permanent breakpoint.

Examples

To re-activate breakpoints using the breakpoints display popup menu:

Change status with a mouse click on this line (menu and highlight do not appear).

Choose this menu item to change the state of the highlighted breakpoint.



To clear a breakpoint

- When displaying memory in mnemonic format, position the mouse pointer over the program line at which you wish to clear a currently set breakpoint (notice the asterisk at the left of the line) and click the *select* mouse button. Or, press and hold the *select* mouse button and choose **Set/Clear Software Breakpoint** from the popup menu.

- When displaying breakpoints, position the mouse pointer over the line displaying the breakpoint you wish to clear, press and hold the *select* mouse button, and choose **Clear (delete) Breakpoint** from the popup menu.

- Place an absolute or symbolic address in the entry buffer; then choose **Breakpoints→Clear ()**.

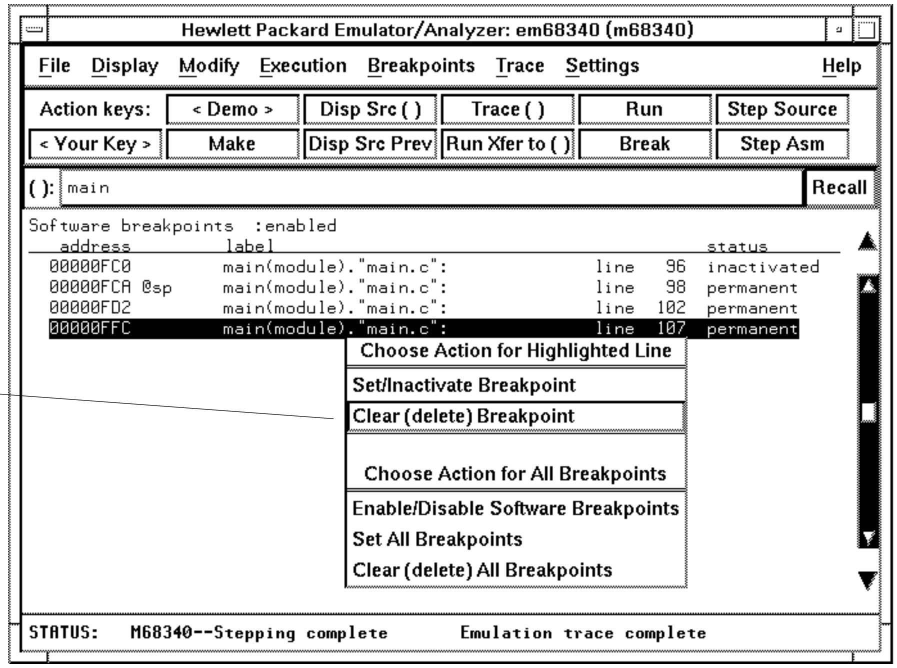
- Using the command line, enter the **modify software_breakpoints clear <address>** command.

When you clear a breakpoint, it is removed from the breakpoints list.

Examples

To clear a software breakpoint using the breakpoints display popup menu:

Bring up the menu and choose this item to clear the highlighted breakpoint.



To clear all breakpoints

- When displaying breakpoints, position the mouse pointer within the Breakpoints Display screen, press and hold the *select* mouse button, and choose **Clear (delete) All Breakpoints** from the popup menu.

- Choose **Breakpoints**→**Clear All**.

- Using the command line, enter the **modify software_breakpoints clear** command.

Displaying and Modifying Registers

This section describes tasks related to displaying and modifying emulation processor registers.

You can display the contents of an individual register or of all the registers. The register classes and names are listed in the following table.

Register Class	**Register**	**Description**
BASIC	PC	Program Counter
	ST	Status Register
(General	USP	User Stack Pointer
Registers)	SSP	Supervisor Stack Pointer
	D0 - D7	Data Registers 0 through 7
	A0 - A7	Address Registers 0 through 7
	VBR	Vector Base Register
	SFC, DFC	Alternate Function Code Registers

Register Class	Register	Description
SIM	MBAR	Module Base Address Register
	SIM_MCR	Module Configuration Register
(System	SYNCR	Clock Synthesizer Control Register
Integration	AVR	Autovector Register
Module)	RSR	Reset Status Register
	PORTA	Port A Data
	DDRA	Port A Data Direction
	PPARA1	Port A Pin Assignment 1
	PPARA2	Port A Pin Assignment 2
	PORTB	Port B Data
	PORTB1	Port B Data
	DDRB	Port B Data Direction
	PPARB	Port B Pin Assignment
	SWIV	Software Interrupt Vector
	SYPCR	System Protection Control
	PICR	Periodic Interrupt Control Register
	PITR	Periodic Interrupt Timing Register
	SWSR	Software Service
	CS0MASK	Address Mask CS0
	CS0ADDR	Base Address CS0
	CS1MASK	Address Mask CS1
	CS1ADDR	Base Address CS1
	CS2MASK	Address Mask CS2
	CS2ADDR	Base Address CS2
	CS3MASK	Address Mask CS3
	CS3ADDR	Base Address CS3
DMA1/2	DMA_MCR1/2	Module Configuration Register
	INTR1/2	Interrupt Register
(DMA	CCR1/2	Channel Control Register
Controller	CSR1/2	Channel Status Register
Modules	FCR1/2	Function Code Register
1 and 2)	SAR1/2	Source Address Register
	DAR1/2	Destination Address Register
	BTC1/2	Byte Transfer Counter

Register Class	Register	Description
SERIAL	SERIAL_MCR	Module Configuration Register
	ILR	Interrupt Level
(Serial Module)	IVR	Interrupt Vector
	MR1A	Mode Register 1A
	SRA	Status Register A
	CSRA	Clock-Select Register A
	CRA	Command Register A
	RBA	Receiver Buffer A
	TBA	Transmitter Buffer A
	IPCR	Input Port Change Register
	ACR	Auxiliary Control Register
	ISR	Interrupt Status Register
	IER	Interrupt Enable Register
	MR1B	Mode Register 1B
	SRC	Status Register B
	CSRB	Clock-Select Register B
	CRB	Command Register B
	RBB	Receiver Buffer B
	TBB	Transmitter Buffer B
	IP	Input Port Register
	OPCR	Output Port Control Register
	OP_SET	Output Port Bit Set
	OP_RST	Output Port Bit Reset
	MR2A	Mode Register 2A
	MR2B	Mode Register 2B
TIMER1/2	TIMER_MCR1/2	Module Configuration Register
	IR1/2	Interrupt Register
(Timer Modules	CR1/2	Control Register
1 and 2)	SR1/2	Status/Prescaler Register
	CNTR1/2	Counter Register
	PREL11/2	Preload 1 Register
	PREL21/2	Preload 2 Register
	COM1/2	Compare Register

Register Class	Register	Description
CF_SIM (Emulator Configuration Registers)	CF_MBAR CF_SIM_MCR CF_PPARA1 CF_PPARA2 CF_CS0MASK CF_CS0ADDR CF_CS1MASK CF_CS1ADDR CF_CS2MASK CF_CS2ADDR CF_CS3MASK CF_CS3ADDR	Module Base Address Register Module Configuration Register Port A Pin Assignment 1 Port A Pin Assignment 2 Address Mask CS0 Base Address CS0 Address Mask CS1 Base Address CS1 Address Mask CS2 Base Address CS2 Address Mask CS3 Base Address CS3

To display register contents

- Choose **Display→Registers**.
- Using the command line, enter the **display registers** command.

When displaying registers, you can display classes of registers and individual registers.

The least significant bit of the emulation processor's MBAR register must be a 1 (which means the MBAR contents are valid) before you can display or modify registers in the SIM, DMA1, DMA2, SERIAL, TIMER1, or TIMER2 register classes.

The contents of write-only registers cannot be displayed.

Examples

To display the basic register contents:

display registers `<RETURN>`

To display the SIM module configuration register:

modify register SIM MBAR to `100001h <RETURN>`

display registers SIM SIM_MCR `<RETURN>`

To display the contents of the emulator configuration registers (CF_SIM class):

display registers CF_SIM `<RETURN>`

To modify register contents

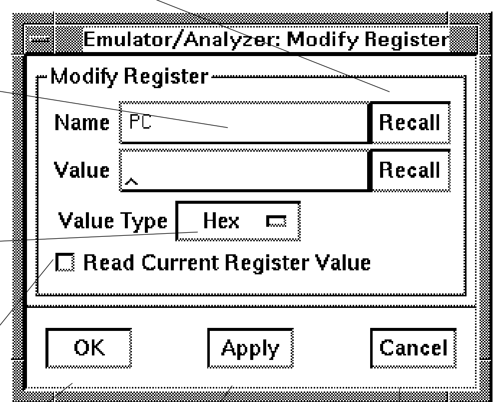
- Choose **Modify→Registers...** and use the dialog box to name the register and specify its value.

Clicking the "Recall" pushbutton lets you select register names and values from predefined or previously specified entries.

Placing the mouse pointer in the text entry area lets you type in the register name and value.

To define the type of value, press and hold the *command select* mouse button and drag the mouse to select the value type.

Clicking this checkbox causes the current value of the named register to be placed in the "Value" text entry area.



Clicking this button modifies the register to the value specified and closes the dialog box.

Clicking this button modifies the register to the value specified and leaves the dialog box open.

Clicking this button cancels modification and closes the dialog box.

- Using the command line, enter the **modify register <register> to <value>** command.

  You can modify the emulator configuration registers (CF_SIM register class); however, any changes you make this way are not saved when you save the configuration. You must modify the configuration to have emulator configuration register changes saved.

Displaying and Modifying Memory

You can display and modify the contents of memory in hexadecimal formats and in real number formats. You can also display the contents of memory in assembly language mnemonic format.

This section shows you how to:

- Display memory.
- Display memory in mnemonic format.
- Display memory in mnemonic format at the current PC.
- Return to the previous mnemonic display.
- Display memory in hexadecimal format.
- Display memory in real number format.
- Display memory at an address.
- Display memory repetitively.
- Modify memory.
- Modify memory at an address.

To display memory

- Choose **Display→Memory**.

This command either re-displays memory in the format specified by the last memory display command, or, if no previous command has been executed, displays memory as hexadecimal bytes beginning at address zero.

To display memory in mnemonic format

- To display memory at a particular address, place an absolute or symbolic address in the entry buffer; then, choose **Display→Memory→Mnemonic ()**, or, using the command line, enter the **display memory <address> mnemonic** command.

- To display memory at the current program counter address, choose **Display→Memory→Mnemonic at PC**, or, using the command line, enter the **display memory mnemonic at_pc** command.

A highlighted bar shows the location of the current program counter address. This allows you to view the program counter while stepping through user program execution.

Whether source lines, assembly language instructions, or symbols are included in the display depends on the modes you choose with the **Settings→Source/Symbols Modes** or **Settings→Display Modes** pulldown menu items. See the "Changing the Interface Settings" section.

If symbols are loaded into the interface, the default is to display source only.

To return to the previous mnemonic display

- Choose **Display→Memory→Mnemonic Previous**.

- Using the command line, enter the **display memory mnemonic previous_display** command.

This command is useful for quickly returning to the previous mnemonic memory display.

For example, suppose you are stepping source lines and you step into a function that you would like to step over. You can return to the previous mnemonic memory display, set a breakpoint at the line following the function call, and run the program from the current program counter.

To display memory in hexadecimal format

- Place an absolute or symbolic address in the entry buffer; then, choose **Display→Memory→Hex ()** and select the size from the cascade menu.

- Using the command line, enter the **display memory <address> blocked <size>** command.

  This command displays memory as hexadecimal values beginning at the address in the entry buffer.

Examples

To display memory in absolute word format:

display memory `ascii_old_data` ***absolute words*** `<RETURN>`

```
Memory  :@sp :words :absolute :update
  address  label             data  :hex      :ascii
 000072DA  _ascii_old_d            2020
 000072DC                          2020
 000072DE                          2034          4
 000072E0                          3800        8.
 000072E2                          2350        #P
 000072E4                          6173        as
 000072E6                          2020
 000072E8                          3400        4.
 000072EA                          2020
 000072EC                          3736        76
 000072EE                          2E32        .2
 000072F0                          3200        2.
 000072F2                          2353        #S
 000072F4                          7769        wi
 000072F6                          2020
 000072F8                          3100        1.
 000072FA                          004C        .L
```

To display memory in blocked byte format:

display memory `ascii_old_data` ***blocked bytes*** `<RETURN>`

```
Memory  :@sp :bytes :blocked :update
  address       data       :hex                                          :ascii
 000072DA-E1    20   20   20   20   20   34   38   00                      4 8 .
 000072E2-E9    23   50   61   73   20   20   34   00        # P a s         4 .
 000072EA-F1    20   20   37   36   2E   32   32   00              7 6   . 2 2 .
 000072F2-F9    23   53   77   69   20   20   31   00        # S w i         1 .
 000072FA-01    00   4C   45   41   52   45   44   00        . L E A   R E D .
 00007302-09    4C   65   6E   20   20   20   31   00        L e n           1 .
 0000730A-11    43   4C   45   41   52   45   44   00        C L E A   R E D .
 00007312-19    41   76   65   20   30   2E   30   00        A v e     0 . 0 .
 0000731A-21    43   4C   45   41   52   45   44   00        C L E A   R E D .
 00007322-29    43   4C   45   41   52   45   44   00        C L E A   R E D .
 0000732A-31    43   4C   45   41   52   45   44   00        C L E A   R E D .
 00007332-39    43   4C   45   41   52   45   44   00        C L E A   R E D .
 0000733A-41    43   4C   45   41   52   45   44   00        C L E A   R E D .
 00007342-49    43   4C   45   41   52   45   44   00        C L E A   R E D .
 0000734A-51    43   4C   45   41   52   45   44   00        C L E A   R E D .
 00007352-59    43   4C   45   41   52   45   44   00        C L E A   R E D .
 0000735A-61    43   4C   45   41   52   45   44   00        C L E A   R E D .
```

To display memory in real number format

- Place an absolute or symbolic address in the entry buffer; then, choose **Display→Memory→Real ()** and select the size from the cascade menu.

- Using the command line, enter the **display memory <address> real <size>** command.

Displays memory as a list of real number values beginning at the address in the entry buffer. Short means four byte real numbers and long means eight byte real numbers.

Examples

To display memory in 64-bit real number format:

display memory real long <RETURN>

```
Memory  :@sp :long real :update
  address  label              data  :real
 000072DA  _ascii_old_d      6.01347001874370E-154
 000072E2                    1.37554080117403E-138
 000072EA                    6.04709978053044E-154
 000072F2                    1.63466481035068E-138
 000072FA                    3.14520063816813E-307
 00007302                    1.07615524484636E+060
 0000730A                    1.59148924067205E+016
 00007312                    2.34828830112762E+007
 0000731A                    1.59148924067205E+016
 00007322                    1.59148924067205E+016
 0000732A                    1.59148924067205E+016
 00007332                    1.59148924067205E+016
 0000733A                    1.59148924067205E+016
 00007342                    1.59148924067205E+016
 0000734A                    1.59148924067205E+016
 00007352                    1.59148924067205E+016
 0000735A                    1.59148924067205E+016
```

To display memory at an address

- Place an absolute or symbolic address in the entry buffer; then, choose **Display→Memory→At ()**.

This command displays memory in the same format as that of the last memory display command. If no previous command has been issued, memory is displayed as hexadecimal bytes.

To display memory repetitively

- Choose **Display→Memory→Repetitively**.

- Using the command line, enter the **display memory repetitively** command.

The memory display is constantly updated. The format is specified by the last memory display command.

This command is ignored if the last memory display command was a mnemonic display.

To modify memory

- Choose **Modify→Memory** and complete the command using the command line.

- To modify memory at a particular address, place an absolute or symbolic address in the entry buffer; then, choose **Modify→Memory at ()** and complete the command using the command line.

- Using the command line, enter the **modify memory** command.

You can modify the contents of one memory location or a range of memory locations. Options allow you to modify memory in byte, short, word, and real number formats.

Displaying Data Values

The data values display lets you view the contents of memory as data types. You can display data values in the following formats:

bytes
8-bit integers
unsigned 8-bit integers
chars
words
16-bit integers
unsigned 16-bit integers
long words
32-bit integers
unsigned 32-bit integers

This section shows you how to:

- Display data values.
- Clear the data values display and add a new item.
- Add item to the data values display.

To display data values

- Choose **Display→Data Values**.

- Using the command line, enter the **display data** command.

Items must be added to the data values display before you can use this command.

The data display shows the values of simple data types in the user program. When the display mode setting turns ON symbols, a label column that shows symbol values is added to the data display.

Step commands and commands that cause the emulator to enter the monitor (for example, encountering a breakpoint) cause the data values screen to be updated.

To clear the data values display and add a new item

- Place an absolute or symbolic address in the entry buffer; then, choose **Display→Data Values→New ()** and select the data type from the cascade menu.
- Using the command line, enter the **display data <address>** command.

To add items to the data values display

- Place an absolute or symbolic address in the entry buffer; then, choose **Display→Data Values→Add ()** and select the data type from the cascade menu.
- Using the command line, enter the **display data , <address>** command.

Changing the Interface Settings

This section shows you how to:

- Set the source/symbol modes.
- Set the display modes.

To set the source/symbol modes

- To display assembly language mnemonics with absolute addresses, choose **Settings→Source/Symbol Modes→Absolute**, or, using the command line, enter the **set source off symbols off** command.

- To display assembly language mnemonics with absolute addresses replaced by global and local symbols where possible, choose **Settings→Source/Symbol Modes→Symbols**, or, using the command line, enter the **set source off symbols on** command.

- To display assembly language mnemonics intermixed with high-level source lines, choose **Settings→Source/Symbol Modes→Source Mixed**, or, using the command line, enter the **set source on symbols on** command.

- To display only high-level source lines, choose **Settings→Source/Symbol Modes→Source Only**, or, using the command line, enter the **set source only symbols on** command.

The source/symbol modes affect mnemonic memory displays and trace displays.

Each display mode cascade menu choice is a toggle. Choosing one of these items causes it to be the only one active and toggles all others off. Provided that symbols were loaded, the interface defaults to:

- Source only for mnemonic memory displays.
- Source mixed for trace listing displays.

To set the display modes

- Choose **Settings→Display Modes...** to open the display modes dialog box.

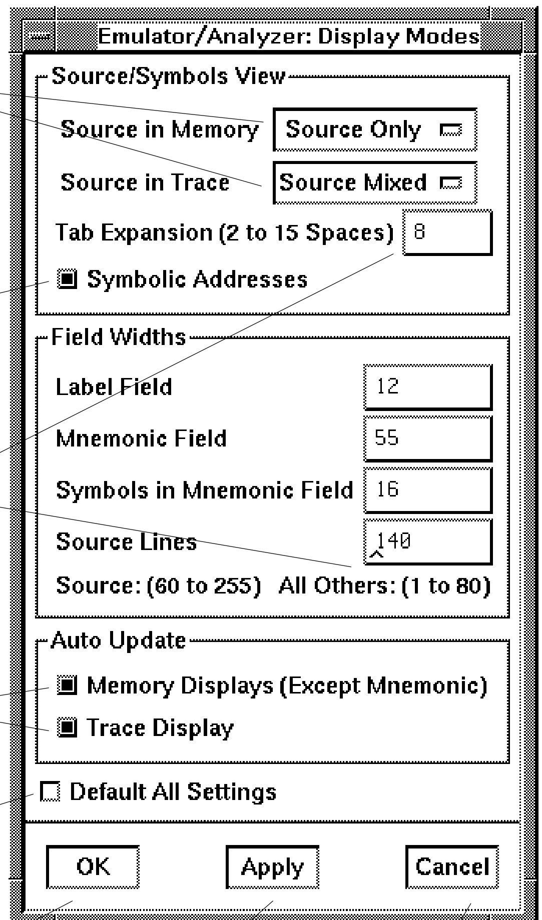
Press and hold the *select* mouse button and drag the mouse to select "Source Only", "Source Mixed", or "Off".

Clicking toggles whether symbolic information is displayed.

Move the mouse pointer to the text entry area and type in the value. Descriptions of the modes follow.

Clicking toggles auto update settings.

Clicking this checkbox changes all display mode settings to their defaults.



Clicking this button saves your changes and closes the dialog box.

Clicking this button saves your changes and leaves the dialog box open.

Clicking this button cancels your changes and closes the dialog box.

Source/Symbols View

Source in Memory specifies whether source lines are included, mixed with assembly code, or excluded from mnemonic memory displays.

Source in Trace specifies whether source lines are included, mixed with stored states, or excluded from trace displays.

Symbolic Addresses specifies whether symbols are included in displays.

Tab Expansion sets the number of spaces displayed for tabs in source lines.

Source/Symbols View

Label Field sets the width (in characters) of the address field in the trace list or label (symbols) field in any of the other displays.

Mnemonic Field sets the width (in characters) of the mnemonic field in memory mnemonic, trace list, and register step mnemonic displays. It also changes the width of the status field in the trace list.

Symbols in Mnemonic Field sets the maximum width of symbols in the mnemonic field of the trace list, memory mnemonic, and register step mnemonic displays.

Source Lines sets the width (in characters) of the source lines in the memory mnemonic display.

Auto Update

Memory Displays toggles whether memory displays are automatically updated after commands that change memory contents or whether you must enter memory display commands to update the display. You may wish to turn off memory display updates, for example, when displaying memory mapped I/O.

Trace Displays toggles whether trace displays are automatically updated when trace measurements complete or whether you must enter trace display commands to update the display. You may wish to turn off trace display updates in one emulator/analyzer window in order to compare the display with a new trace display in another emulator/analyzer window.

Using System Commands

With the system commands, you can:

- Set UNIX environment variables while in the Softkey Interface.
- Display the name of the emulation module.
- Display the event log.
- Display the error log.
- Edit files.
- Copy information to a file or printer.
- Open a terminal emulation window.

To set UNIX environment variables

- Using the command line, enter the **set <VAR>** command.

You can set UNIX shell environment variables from within the Softkey Interface with the **set <environment_variable> = <value>** command.

Examples

To set the PRINTER environment variable to "lp -s":

```
set PRINTER = "lp -s" <RETURN>
```

After you set an environment variable from within the Softkey Interface, you can verify the value of it by entering **!set <RETURN>**.

To display the name of the emulation module

- Using the command line, enter the **name_of_module** command.

While operating your emulator, you can verify the name of the emulation module. This is also the logical name of the emulator in the emulator device file.

Examples

To display the name of your emulation module:

```
name_of_module <RETURN>
```

The name of the emulation module is displayed on the status line.

To display the event log

- Choose **Display→Event Log**.

- Position the mouse pointer on the status line, press and hold the *select* mouse button, and then choose **Display Event Log** from the popup menu.

- Using the command line, enter the **display event_log** command.

The last 100 events that have occurred during the emulation session are displayed.

The status of the emulator and analyzer are recorded in the event log, as well as the conditions that cause the status to change (for example, software breakpoints and trace commands).

To display the error log

- Choose **Display→Error Log**.

- Position the mouse pointer on the status line, press and hold the *select* mouse button, and then choose **Display Error Log** from the popup menu.

- Using the command line, enter the **display error_log** command.

The last 100 error messages that have occurred during the emulation session are displayed.

To edit files

- Choose **File→Edit→File** and use the dialog box to specify the file name.

- To edit a file based on an address in the entry buffer, place an address reference (either absolute or symbolic) in the entry buffer; then, choose **File→Edit→At () Location**.

- To edit a file based on the current program counter, choose **File→Edit→At PC Location**.

- To edit a file associated with a symbol when you are displaying symbols, position the mouse pointer over the symbol, press and hold the *select* mouse button, and choose **Edit File At Symbol** from the popup menu.

- To edit a file when displaying memory in mnemonic format, position the mouse pointer over the line of source where you want to begin the edit, press and hold the *select* mouse button, and choose **Edit Source** from the popup menu.

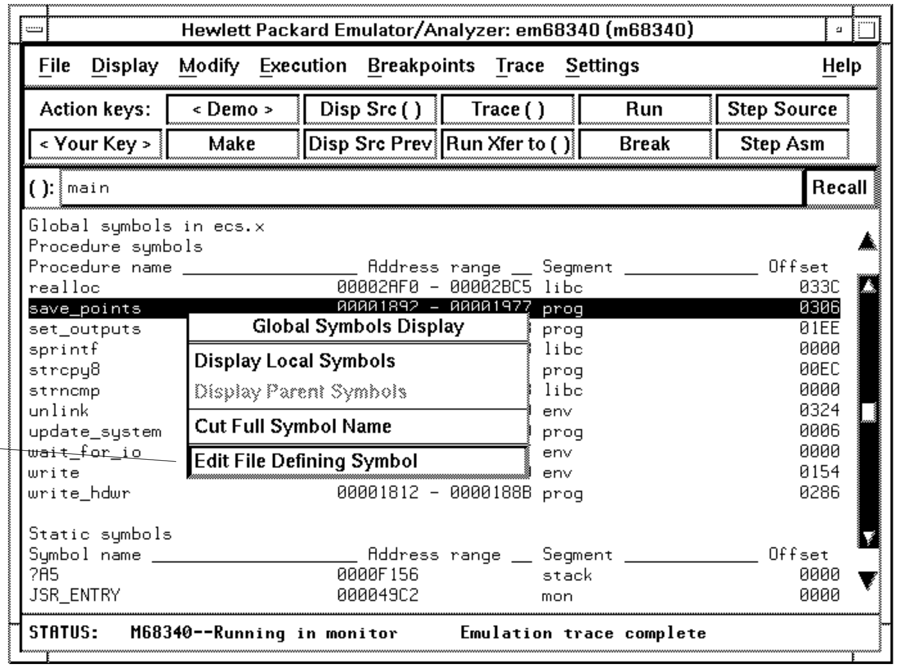
When editing files at addresses, the interface determines which source file contains the code generated for the address and opens an edit session on the file. The interface will issue an error if it cannot find a source file for the address.

The interface will choose the "vi" editor as its default editor, unless you specify another editor by setting an X resource. Refer to the "Setting X Resources" chapter for more information about setting this resource.

You must load symbols before most commands will work because symbol information is needed to be able to locate the files.

Examples

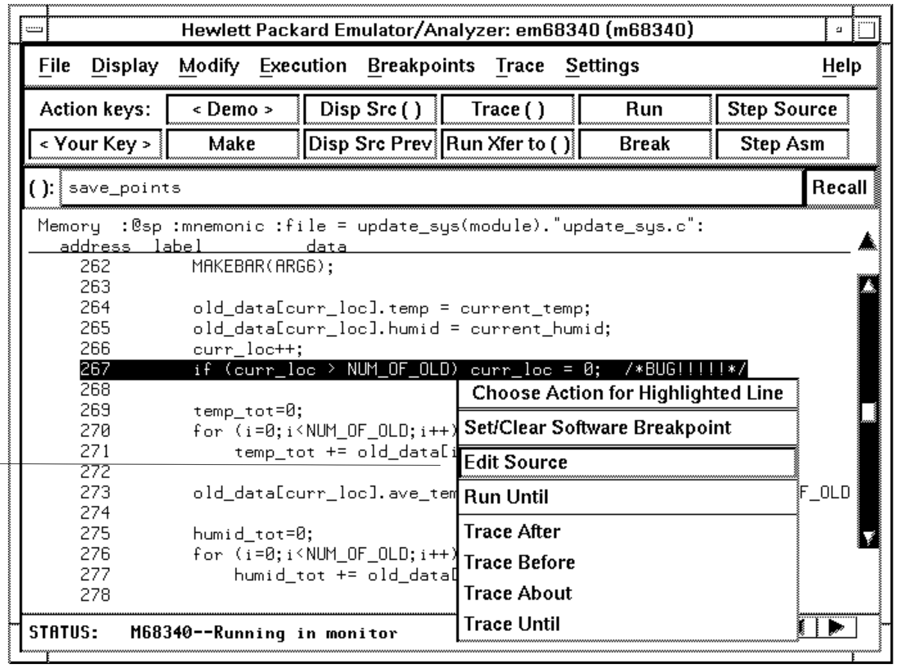
To edit a file that defines a symbol:



Choosing this menu item brings up a terminal window with an edit session open on the file where the highlighted symbol is defined.

To edit a file at a source line:

Choosing this menu item brings up a terminal window with an edit session open on the file where the highlighted source line exists.



To copy information to a file or printer

- Choose **File→Copy**, select the type of information from the cascade menu, and use the dialog box to select the file or printer.

- Using the command line, enter the **copy** command.

ASCII characters are copied to the file or printer.

If you copy information to an existing file, it will be appended to the file.

Refer to the following paragraphs for details about the different copy options.

Display ... Copies information currently in the display area. This option is useful for restricting the number of lines that are copied. Also, this option is useful for copying the contents of register classes other than BASIC.

Memory ... Copies the contents of a range of memory. The format is the same as specified in the last display memory command. For example, if you copy memory after displaying a range of memory in mnemonic format, the file would contain the mnemonic memory information. If there is no previous display memory command, the format used is a blocked hex byte format beginning at address zero.

Data Values ... Copies the contents of the defined data values last displayed. An error occurs if you try to copy data values to a file if you have not yet displayed data values.

Trace ... The most recently captured trace is copied to the file. The copied trace listing is formatted according to the current display mode.

You can set the display mode with the **Settings→Source/Symbols Modes** or **Settings→Display Modes** pulldown menu items. See the "Changing the Interface Settings" section.

Registers ... Copies the current values of the BASIC register class to a file. To copy the contents of the other register classes, first display the registers in that class, and then use the **File→Copy→Display ...** command.

Breakpoints ... Copies the breakpoints list. If no breakpoints are present in the list, only the enable/disable status is copied.

Status ... Copies the emulator/analyzer status display.

Global Symbols ... Copies the global symbols. If symbols have not been loaded, this menu item is grayed-out and unresponsive.

Local Symbols () ... Copies the local symbols from the symbol scope named (by an enclosing symbol) in the entry buffer. If symbols have not been loaded, this menu item is grayed-out and unresponsive.

Pod Commands ... Copies the last 100 lines from the pod commands display.

Error Log ... Copies the last 100 lines from the error log display.

Event Log ... Copies the last 100 lines from event log display.

To open a terminal emulation window

- Choose **File→Term...**

This command opens a terminal window into the current working directory context.

Using Simulated I/O

Simulated I/O is a feature of the emulator/analyzer interface that lets you use the same keyboard and display that you use with the interface to provide input to programs and display program output.

To use simulated I/O, your programs must communicate with the simulated I/O control address and the buffer locations that follow it. (The Hewlett-Packard AxLS compilers, if your program uses I/O, automatically link with environment dependent routines that communicate with the simulated I/O control address and buffer.)

Also, before simulated I/O can work, the emulator must be configured to enable polling of the simulated I/O control address and to define the control address location.

This section shows you how to:

- Display the simulated I/O screen.
- Use simulated I/O keyboard input.

Refer to the *Simulated I/O User's Guide* for complete details on how simulated I/O works.

To display the simulated I/O screen

- Choose **Display**→**Simulated IO**.

Before you can display simulated I/O, polling for simulated I/O must be enabled in the emulator configuration.

Examples

```
Simulated I/O display                                  Status messages disabled
display is open
  53 58 79 88              h    H                    t        T
  53 57 79 87              h   H                     t       T
  52 56 78 86             h   H                     t       T
  52 55 78 85             h  H                      t      T
  51 54 77 84            h  H                      t      T
  51 53 77 83            h H                       t     T
  50 52 76 82           h H                       t     T
  50 50 76 76           H                         T
  49 49 73 75          H                       t T
  48 48 72 74         H                       t T
  48 47 72 73        Hh                       tT
  47 46 71 72       Hh                       tT
  47 45 71 71      H h                       T
  46 44 70 70     H h                       T
  46 43 70 69    H  h                      Tt
  45 42 69 68   H  h                      Tt
```

A message tells you whether the display is open or closed. You can modify the configuration to enable status messages.

To use simulated I/O keyboard input

- To begin using simulated I/O input, choose **Settings→Simulated IO Keyboard**.
- To end simulated I/O and return to using the interface, use the **suspend** softkey.

The command line entry area is used for simulated input with the keyboard. Therefore, if the command line is turned off, choosing this menu item with turn command line display back on.

If you are planning to use even a modest amount of simulated I/O input during an emulation session, it might be a good idea to open another Emulator/Analyzer window to be used exclusively for simulated I/O input and output.

Using Basis Branch Analysis

Basis branch analysis (BBA) is provided by the HP Branch Validator product. This product is used to analyze the testing of your programs, create more complete test suites, and quantify your level of testing.

The HP Branch Validator records branches executed in a program and generates reports that provide information about program execution during testing. It uses a special C preprocessor to add statements that write to a data array when program branches are taken. After running the program in the emulator (using test input), you can store the BBA information to a file. Then, you can generate reports based on the stored information.

This section shows you how to:

- Store BBA data to a file.

Refer to the *HP Branch Validator (BBA) User's Guide* for complete details on the BBA product and how it works.

To store BBA data to a file

- Choose **File→Store→BBA Data** and use the selection dialog box to specify the file name.

The default file name "bbadump.data" can be selected from the dialog box.

7

Using the Emulation Analyzer

Using the Emulation Analyzer

This chapter describes tasks you perform while using the emulation analyzer. These tasks are grouped into the following sections:

- The basics of starting, stopping, and displaying traces.
- Qualifying trigger and store conditions.
- Using the sequencer.
- Modifying trace displays.
- Saving and restoring traces.

The Basics of Starting, Stopping, and Displaying Traces

This section describes the basic tasks that relate to starting and stopping trace measurements.

When you start a trace measurement, the analyzer begins looking at the data on the emulation processor's bus and control signals on each analyzer clock signal. The information seen on a particular clock is called a state.

When one of these states matches the "trigger state" you specify, the analyzer stores states in trace memory. When trace memory is filled, the trace is said to be "complete." The default trigger state specification is "any state," so when you start a trace measurement after initializing the analyzer, the analyzer will "trigger" on the first state it sees and store the following states in trace memory.

Once you start a trace measurement, you can view the progress of the measurement by displaying the trace status.

In some situations, for example, when the trigger state is never found or when the analyzer hasn't filled trace memory, the trace measurement does not complete. In these situations, you can halt the trace measurement.

Once a trace is displayed, you can use the cursor keys and other keys to position the trace list on the display. To speed up the display of traces, you can reduce the depth of the trace list. Also, when entering trace commands, there is a special command that allows you to recall and modify the last trace command entered.

This section describes how to:

- Start trace measurements.
- Display the trace status.
- Stop trace measurements.
- Display the trace.
- Position the trace display on the screen.
- Change the trace depth.
- Modify the last trace command entered.

To start a trace measurement

- Choose **Trace→Everything**.

- Using the command line, enter the **trace** command.

The **trace** command tells the analyzer to begin monitoring the states which appear on the trace signals. You will see a message that confirms that a trace is started.

The default trace command (simply **trace** with no options) will trigger on any state, store all captured states.

Examples

While the emulator is running the user program, you can start the default trace measurement with the command:

`trace` `<RETURN>`

A message is displayed on the status line to show you that the "Emulation trace [has] started", and another message will show you when the "Emulation trace [is] complete".

To display the trace status

- Choose **Display→Status**.

- Using the command line, enter the **display status** command.

In addition to the analyzer information shown on the status line (Emulation trace started, Emulation trace complete, etc.), you can display complete analyzer status with the command below.

Examples

To display the trace status:

display status <RETURN>

```
Status
--------------------------------------------------------------------------
Emulator Status

  M68340--Running user program

Trace Status

  Emulation trace complete
  Arm ignored
  Trigger in memory
  Arm to trigger ?
  States 512 (512) 0..511
  Sequence term 2
  Occurrence left 1
```

The first line of the emulation trace status display shows the user trace has been "completed"; other possibilities are that the trace is still "running" or that the trace has been "halted".

The second line of the trace status display contains information on the arm condition. If the analyzer is always armed, the message "Arm ignored" is displayed. When the arm condition is ignored, the "Arm to trigger" time is not meaningful and a question mark is displayed. (The "Making Coordinated Measurements" chapter explains arm conditions.)

If the analyzer is to be armed by one of the internal signals, either the message "Arm not received" or "Arm received" is displayed. The display indicates if the arm condition happened any time since the most recent trace started, even if it happened after the trace was halted or became complete.

The "Arm to trigger" line displays the amount of time between the arm condition and the trigger. The time displayed will be from -0.04 microseconds to 41.943 milliseconds, less than -0.04 microseconds, or greater than 41.943 milliseconds. If the arm signal is ignored or the trigger is not in memory, a question mark (?) is displayed.

The "States" line shows the number of states that have been stored (out of the number that is possible to store) and the line numbers that the stored states occupy. (The trigger state is always stored on line 0.)

The "Sequence term" line of the trace status display shows the number of the term the sequencer was in when the trace completed. Because a branch **out of the last sequence term** constitutes the trigger, the number displayed is what would be the next term (2 in the preceding example) even though that term is not defined. If the trace is halted, the sequence term number just before the halt is displayed; otherwise, the current sequence term number is displayed. If the current sequence term is changing too quickly to be read, a question mark (?) is displayed.

The "Occurrence left" line of the trace status display shows the number of occurrences remaining before the primary branch can be taken out of the current sequence term. If the occurrence left is changing too quickly to be read, a question mark (?) is displayed.

To stop a trace measurement

- Choose **Trace→Stop**.

- Using the command line, enter the **stop_trace** command.

You can, and most likely will, specify traces whose trigger or storage states are never found. When this happens, the "Emulation trace complete" message is never shown, and the trace continues to run ("Emulation trace running"). When these situations occur, you can halt the trace measurement with the **stop_trace** command.

The **stop_trace** command is also useful to deactivate signals which are driven when the trigger is found (refer to the "Making Coordinated Measurements" chapter).

Examples

To halt a trace measurement:

stop_trace <RETURN>

When the **stop_trace** command is entered, the message "Emulation trace halted" is displayed.

To display the trace

- Choose **Trace→Display** or **Display→Trace**.

- Using the command line, enter the **display trace** command.

You can display captured trace data with the **display trace** command. The available options to the **display trace** command are described in the "Modifying the Trace Display" section later in this chapter.

Examples

To display the trace:

display trace <RETURN>

```
Trace List              Offset=0                             More data off screen
Label:      Address           Opcode or Status w/ Source Lines        time count
Base:       symbols                  mnemonic w/symbols                relative
after  strcpy8+00000012  ADDQ.L    #1,D3                             ------------
+001   strcpy8+00000014  NOP                                          720     nS
+002   strcpy8+00000016  CMP.L     D4,D3                              680     nS
+003   strcpy8+00000018  BLT.B     prog|strcpy8+$0010                 720     nS
+004   strcpy8+0000001A  NOP                                          720     nS
+005   strcpy8+00000010  NOP                                          720     nS
+006   strcpy8+00000012  ADDQ.L    #1,D3                              720     nS
+007   strcpy8+00000014  NOP                                          720     nS
+008   strcpy8+00000016  CMP.L     D4,D3                              720     nS
+009   strcpy8+00000018  BLT.B     prog|strcpy8+$0010                 720     nS
+010   strcpy8+0000001A  NOP                                          680     nS
+011   strcpy8+00000010  NOP                                          720     nS
+012   strcpy8+00000012  ADDQ.L    #1,D3                              720     nS
+013   strcpy8+00000014  NOP                                          720     nS
+014   strcpy8+00000016  CMP.L     D4,D3                              720     nS
+015   strcpy8+00000018  BLT.B     prog|strcpy8+$0010                 720     nS
```

The first column in the trace list contains the line number. The trigger is always on line 0.

The second column contains the address information associated with the trace states. Addresses in this column may be locations of instruction opcodes on fetch cycles, or they may be sources or destinations of operand cycles.

The third column shows mnemonic information about the emulation bus cycle.

The fourth column shows the count information (time is counted by default). "Relative" indicates that each count is relative to the previous state.

You can use the scrollbar in the Graphical User Interface or the <NEXT> and <PREV> keys in the Softkey Interface to scroll through the trace list a page at a time. The <Up arrow> and <Down arrow> keys will scroll through the trace list a line at a time. You can also display the trace list centered around a specific line number (for example, **display trace 100 <RETURN>**). Refer to the "Modifying the Trace Display" section for more information on the trace list display.

Note that when a trigger condition is found but not enough states are captured to fill trace memory, the status line will show the trace is still running. You can display all but the last captured state in this situation; you must halt the trace to display the last captured state.

To position the trace display on screen

- Use the scroll bar or the <Up arrow>, <Down arrow>, <PREV>, <NEXT>, <CTRL>f, and <CTRL>g keys.

The trace display command can display up to 1024 states, not all of which can appear on the screen at the same time. However, you can reposition the display on the screen with the keys described below.

The <Up arrow> and <Down arrow> (or roll up and roll down) keys move the display up or down on the screen one line at a time.

The <PREV> and <NEXT> (or page up and page down) keys allow you to move the display up or down a page at a time.

The <CTRL>f and <CTRL>g keys allow you to move the display left or right, respectively. These keys are used when the width of the address or mnemonic/absolute columns is increased so that not all the trace display data can be displayed across the screen.

To change the trace depth

- Using the command line, enter the **display trace depth** command.

The **display trace depth** command allows you to specify the number of states that are displayed. By reducing the trace depth, you can shorten the time it takes for the interface to upload the trace information. You can increase the trace depth to view more states of the current trace.

The maximum number of trace states is 1024 when counting is turned off, 512 otherwise. The minimum trace depth is 9.

If you wish to reduce the number of states that are displayed, the **display trace depth** command must be entered before the **trace** command. You cannot use this command to reduce the number of states displayed in the current trace.

To modify the last trace command entered

- Choose **Trace**→**Trace Spec** and use the dialog box to select and edit a trace command.

- Using the command line, enter the **trace modify_command** command.

The Trace Specification Selection dialog box contains a list of trace specifications executed during the emulation session as well as any predefined trace specifications present at interface startup.

You can predefine trace specifications and set the maximum number of entries for the dialog box by setting X resources (see the "Setting X Resources" chapter).

The **trace modify_command** command recalls the last trace command. The advantage of this command over command recall is that you do not have to move forward and backward over other commands to find the last trace command; also, the last trace command is always available, no matter how many commands have since been entered.

Qualifying Trigger and Store Conditions

This section describes tasks relating to the qualification of trigger and storage states.

You can trigger on, or store, specific states or specific values on a set of trace signals (which are identified by trace labels).

Also, you can *prestore* states. The prestore qualifier is a second storage qualifier used for storing states that occur before the normally stored states. Prestore is useful for capturing entry points to procedures or for identifying where global variables are accessed from.

This section describes how to:

- Qualify the trigger state and its position in the trace.
- Trigger on a number of occurrences of some state.
- Qualify states stored in the trace.
- Qualify prestore states.
- Change the count qualifier.
- Trace until the analyzer is halted.
- Cause the emulator to break into the monitor when the analyzer triggers.

Expressions in Trace Commands

When modifying the analysis specification, you can enter expressions which consist of values, symbols, and operators.

Values Values are numbers in hexadecimal, decimal, octal, or binary. These number bases are specified by the following characters:

B b	Binary (example: 10010110b).
Q q O o	Octal (example: 377o or 377q).
D d (default)	Decimal (example: 2048d or 2048).

H h	Hexadecimal (example: 0a7fh). You must precede any hexadecimal number that begins with an A, B, C, D, E, or F with a zero.

Don't care digits may be included in binary, octal, or hexadecimal numbers and they are represented by the letters **X** or **x**. A zero must precede any numerical value that begins with an "X".

Symbols A symbol database is built when the absolute file is loaded into the emulator. Both global and local symbols can be used when entering expressions. Global symbols are entered as they appear in the global symbols display. When specifying a local symbol, you must include the name of the module ("anly.c") as shown below.

```
anly.c:cmp_function
```

Operators Analysis specification expressions may contain operators. All operations are carried out on 32-bit, two's complement integers. (Values which are not 32 bits will be sign extended when expression evaluation occurs.)

The available operators are listed below in the order of evaluation precedence. Parentheses are also allowed in expressions to change the order of evaluation.

-, ~	Unary two's complement, unary one's complement. The unary two's complement operator is not allowed on constants containing don't care bits.
*****, /, **%**	Integer multiply, divide, and modulo. These operators are not allowed on constants containing don't care bits.
+, **-**	Addition, subtraction. These operators are not allowed on constants containing don't care bits.
&	Bitwise AND.
|	Bitwise inclusive OR.

Values, symbols, and operators may be used together in analysis specification expressions. For example, if the local symbol exists, the following is a valid expression:

```
module.c:symb+0b67dh&0fff00h
```

However, you cannot add two symbols unless one of them is an EQU type symbol.

Emulation Analyzer Trace Signals

When you qualify states, you specify values that should be found on the analyzer trace signals. The emulation analyzer trace signals are described in the table that follows.

Emulation Analyzer Trace Signals		
Trace Signals	**Signal Name**	**Signal Description**
0-31	A0-A31	Address Lines 0-31.
32-47	D0-D15	Data Lines 0-15.
64	BKG_L	Background Debug Mode (BDM) active. This signal is used to qualify the analyzer clock for tracing only foreground or only background cycles.
65 66 67	FC0 FC1 FC2	Function Codes 0-2. These lines to the analyzer are derived from the 68340 processor’s function code lines. The function code meanings are: 001 - User Data Space 010 - User Program Space 101 - Supervisor Data Space 110 - Supervisor Program Space 111 - CPU Space
68	R/*W	Read/write signal.
69 70	SIZ0 SIZ1	Number of bytes remaining to be transferred.
71	CS_BYTE_L	Chip select byte/word signal.
72 73	DS0_L DS1_L	Data size acknowledge. Note that the 68340 SIM can be programmed to internally generate the DSACKx signals for external accesses; in this case, the DSACKx values do not show up on these trace signals.
74	BERR_L	Bus error active.
75	HALT_L	Halt active.
76	CODE_L	Instruction execution active.
77	FLUSH_L	Instruction pipeline flush active.
78	FC3	Function code 3. This can be set by the 68340 DMA controller for DMA transfers; however
79	CS0_L	Chip select 0 active.

State Qualifiers

Whenever a state can be specified in the trace command (trigger state, storage state, prestore state, etc.), you will see the following softkeys that allow you to qualify the state:

address	The value following this softkey is searched for on the lines that monitor the emulation processor’s address bus.
data	The value following this softkey is searched for on the lines that monitor the emulation processor’s data bus.
status	The value following this softkey is searched for on the lines that monitor other emulation processor signals.

When a value is specified without one of these softkeys it is assumed to be an address value.

Predefined Values for Qualifiers When you specify status qualifiers for analyzer states (by pressing the **status** softkey), you will be given the following softkeys which are predefined values for the qualifiers.

Predefined Equates for Analyzer Status

Equate	Value	Description
buserror	0xxxx x0xx xxxx xxxxb	/BERR active
code	0xxx0 xxxx xxxx xxxxb	code execution cycles
code_tfr	0xx00 xxxx xxxx xxxxb	first instruction following a pipeline flush
cpu	0xxxx xxxx xxxx 111xb	CPU space function code
csx_byte	0xxxx xx11 0xxx xxxxb	byte data transfer, chip select active ($\overline{\text{DSACKx}}$ not internally generated)
csx_word	0xxxx xx11 1xxx xxxxb	word data transfer, chip select active ($\overline{\text{DSACKx}}$ not internally generated)
data	0x0xx xxxx xxxx x01xb	data cycle
dma	0x1xx xxxx xxxx xxxxb	DMA space function code (if used by DMA controller module)
ds_byte	0xxxx xx10 xxxx xxxxb	byte data transfer
ds_word	0xxxx xx01 xxxx xxxxb	word data transfer
memread	0xxxx xxxx xxx1 xxxxb	memory read
memwrite	0xxxx xxxx xxx0 xxxxb	memory write
prog	0x0xx xxxx xxxx x10xb	program space function code
rerun	0xxxx 00xx xxxx xxxxb	/BERR and /HALT active (retry)
siz_3byt	0xxxx xxxx x11x xxxxb	3 byte access
siz_byte	0xxxx xxxx x01x xxxxb	byte access
siz_long	0xxxx xxxx x00x xxxxb	long word access
siz_word	0xxxx xxxx x10x xxxxb	word access
super	0x0xx xxxx xxxx 1xxxb	supervisor space function code
supdata	0x0xx xxxx xxxx 101xb	supervisor data space function code
supprog	0x0xx xxxx xxxx 110xb	supervisor program space function code
user	0x0xx xxxx xxxx 0xxxb	user space function code
userdata	0x0xx xxxx xxxx 001xb	user data space function code
userprog	0x0xx xxxx xxxx 010xb	user program space function code

These predefined values may be used as other values would be used. For example:

trace after status write

is the same as:

trace after status `0xxxxxxxxxxx0xxxxb`

To qualify the trigger state and position

- Enter a trigger state specification in the entry buffer; then, choose **Trace→After ()**, **Trace→About ()**, or **Trace→Before ()**.

- When displaying memory in mnemonic format, position the mouse pointer over the source line where you want to set the trace trigger, press and hold the *select* mouse button and choose **Trace After**, **Trace Before**, or **Trace About** from the popup menu.

- Using the command line, enter the **trace after**, **trace about**, or **trace before** commands.

Tracing after the trigger state says states that occur after the trigger state should be saved; in other words, the trigger is positioned at the top of the trace.

Tracing before the trigger state says states that occur before the trigger state should be saved; in other words, the trigger is positioned at the bottom of the trace.

Tracing about the trigger state says states that occur before and after the trigger state should be saved; in other words, the trigger is positioned at the center of the trace.

When the analyzer counts time or states, the actual trigger position is within +/- 1 state of the number specified. When counts are turned OFF, the actual trigger position is within +/- 3 states of the number specified.

Usually, when you enter a **trace about** command, the trigger state (line 0) is labeled "about". However, if there are three or fewer states before the trigger, the trigger state is labeled "after". Likewise, if there are 3 or fewer states after the trigger, the trigger state is labeled "before".

The state you define after **trace after**, **trace about**, or **trace before** is the state that will trigger the analyzer and cause states to be stored.

Examples

Suppose you want to look at the execution of the demo program after the call of the "update_system()" function (main.c: line 102) occurs. To trigger on this address, enter:

trace after address `main."main.c": line 102 <RETURN>`

set source on inverse_video on symbols on `<RETURN>`

display trace `<RETURN>`

```
Trace List                 Offset=0                          More data off screen
Label:      Address           Opcode or Status w/ Source Lines          time count
Base:       symbols                  mnemonic w/symbols                  relative
      ##########main.c - line   101 thru   102 ##################################
          {
              update_system();
after   pr|main+00000012  JSR      up.update_system                     720     nS
+001    pr|main+00000014   $0000         supr prgm word rd (ds16)       720     nS
+002    pr|main+00000016   $1592         supr prgm word rd (ds16)       720     nS
      ##########main.c - line   103 #############################################
              num_checks++;
+003    pr|main+00000018  incomplete instr.: /52B9/????/                720     nS
+004    sysstac+00007F90   $0000         supr data long wr (ds16)       720     nS
+005    sysstac+00007F92   $0FD8         supr data word wr (ds16)       720     nS
      ##########update_sys.c - line     1 thru    47 ###########################
      #include <stdio.h>

      void
      update_system()
```

In the preceding trace list, line 0 (labeled "after") shows the beginning of the program loop.

To trigger on a number of occurrences of some state

- Use the **occurs <#TIMES>** after specifying the trigger state.

When specifying a trigger state, you can include an occurrence count. The occurrence count specifies that the analyzer trigger on the Nth occurrence of some state.

The default base for an occurrence count is decimal. You may specify occurrence counts from 1 to 65535.

Examples

To trigger on the 20th occurrence of the call of the "update_system()" function (main.c: line 102):

```
trace after address main."main.c": line 102 occurs 20
<RETURN>
```

To qualify states stored in the trace

- Enter a storage state specification in the entry buffer; then, choose **Trace→Only ()**.

- Using the command line, use the **only** option in the **trace** command.

By default, all captured states are stored; however, you can qualify which states get stored by using the **trace** command’s **only** option.

Examples

When the emulator is running the demo program, to store *only* accesses of the "target_temp" variable:

trace after `main."main.c": line 102`
only `target_temp <RETURN>`

```
Trace List                   Offset=0                           More data off screen
Label:      Address            Opcode or Status w/ Source Lines         time count
Base:       symbols                    mnemonic w/symbols                relative
          ##########main.c - line   101 thru   102 ##################################
              {
                 update_system();
after     pr|main+00000012  incomplete instr.: /4EB9/????/              ------------
+001      dat|_target_temp   $0048         supr data word rd (ds16)        3.50  mS
+002      dat|_target_temp   $0047         supr data word wr (ds16)      800     nS
+003      dat|_target_temp   $0047         supr data word rd (ds16)        2.2   uS
+004      dat|_target_temp   $0047         supr data word rd (ds16)        4.62  mS
+005      dat|_target_temp   $0047         supr data word rd (ds16)       51.6   mS
+006      dat|_target_temp   $0047         supr data word rd (ds16)        5.84  uS
+007      dat|_target_temp   $0047         supr data word rd (ds16)        2.8   uS
+008      dat|_target_temp   $0047         supr data word rd (ds16)        2.9   uS
+009      dat|_target_temp   $0047         supr data word rd (ds16)       47.4   mS
+010      dat|_target_temp   $0046         supr data word wr (ds16)      840     nS
+011      dat|_target_temp   $0046         supr data word rd (ds16)        2.1   uS
+012      dat|_target_temp   $0046         supr data word rd (ds16)        4.62  mS
```

Notice the trigger state (line 0, labeled "after") is included in the trace list; trigger states are always stored.

To prestore states before qualified store states

- Enter a storage state specification in the entry buffer; then, choose **Trace→Only () Prestore**.

- Use the **prestore** option in the **trace** command.

Prestore allows you to save up to two states which precede a normal store state. Prestore is turned off by default. However, you can use the **trace** command's **prestore** option to specify a prestore qualifier.

Prestore is useful when you want to find the cause of a particular state. For example, if a variable is accessed from many different places in the program, you can qualify the trace so that only accesses of that variable are stored. Then, you can turn on prestore to find out where accesses of that variable originate from.

States which satisfy the prestore qualifier and the storage qualifier at the same time are stored as normal states.

Examples

To storing only write accesses to the variable "target_temp" and prestore the two previous states:

trace after `main."main.c": line 102`
only `target_temp` ***status memwrite***
prestore anything `<RETURN>`

```
Trace List                Offset=0                    More data off screen
Label:       Address           Opcode or Status w/ Source Lines       time count
Base:        symbols                  mnemonic w/symbols               relative
      ##########main.c - line   101 thru   102 ##################################
                {
                    update_system();
after  pr|main+00000012  incomplete instr.: /4EB9/????/               ------------
      ##########update_sys.c - line   105 ######################################
                    if (*temperature <= MIN_TEMP) temp_dir = up;
pstore get_tar+0000004E  incomplete instr.: /0C53/????/
pstore dat|_target_temp   $004A          supr data word rd (ds16)
+003   dat|_target_temp   $0049          supr data word wr (ds16)        3.50  mS
      ##########update_sys.c - line   105 ######################################
                    if (*temperature <= MIN_TEMP) temp_dir = up;
pstore get_tar+0000004E  incomplete instr.: /0C53/????/
pstore dat|_target_temp   $0049          supr data word rd (ds16)
+006   dat|_target_temp   $0048          supr data word wr (ds16)      146.    mS
      ##########update_sys.c - line   105 ######################################
                    if (*temperature <= MIN_TEMP) temp_dir = up;
```

To change the count qualifier

- Use the **counting** option in the **trace** command.

After initializing the analyzer, the default count qualifier is "time", which means that the time between states is saved. When time is counted, up to 512 states can be stored in the trace.

When you count states, the counter is incremented each time the state is captured (not necessarily stored) by the analyzer. When a state is counted, up to 512 states can be stored in the trace.

When you turn OFF counting, up to 1024 states can be stored in the trace.

Examples

Suppose you want to know how many loops of the program occur between calls of the "do_sort" function. To change the count qualifier to count a state that occurs once for each loop of the program, enter:

trace only `do_sort`
counting state `main."main.c": line 102 <RETURN>`

set source off `<RETURN>`

```
Trace List                  Offset=0
Label:      Address                    Opcode or Status                  state count
Base:       symbols                    mnemonic w/symbols                  relative
after  write_h+00000022 CMP.L    A0,D4                                  ------------
+001   pro|main.do_sort  incomplete instr.: /4E56/????/                            1
+002   pro|main.do_sort  incomplete instr.: /4E56/????/                            4
+003   pro|main.do_sort  incomplete instr.: /4E56/????/                            4
+004   pro|main.do_sort  incomplete instr.: /4E56/????/                            4
+005   pro|main.do_sort  incomplete instr.: /4E56/????/                            4
+006   pro|main.do_sort  incomplete instr.: /4E56/????/                            4
+007   pro|main.do_sort  incomplete instr.: /4E56/????/                            4
+008   pro|main.do_sort  incomplete instr.: /4E56/????/                            4
+009   pro|main.do_sort  incomplete instr.: /4E56/????/                            4
+010   pro|main.do_sort  incomplete instr.: /4E56/????/                            4
+011   pro|main.do_sort  incomplete instr.: /4E56/????/                            4
+012   pro|main.do_sort  incomplete instr.: /4E56/????/                            4
+013   pro|main.do_sort  incomplete instr.: /4E56/????/                            4
+014   pro|main.do_sort  incomplete instr.: /4E56/????/                            4
+015   pro|main.do_sort  incomplete instr.: /4E56/????/                            4
```

The trace listing above shows that the program loops 4 times for each call of the "do_sort" function.

To trace until the analyzer is halted

- Choose **Trace→Until Stop**.
- Using the command line, enter the **trace on_halt** command.

The **trace on_halt** command allows you to prevent triggering. In other words, the trace runs until you enter the **stop_trace** command. The **trace on_halt** command is the same as tracing **before** a state that never occurs.

The **trace on_halt** command is useful, for example, when you wish to trace the states leading up to a break into the monitor. Suppose your program breaks on an access to guarded memory. To trace the states that lead up to the break, enter the **trace on_halt** command, and run the program. When the break occurs, the emulator is running in the background monitor, and the analyzer is no longer capturing states. To display the states leading up to the break, enter the **stop_trace** command (and the **display trace** command if traces are not currently being displayed).

When the **on_halt** option is used in a trace command, the trigger condition (and position) options, as well as the **repetitively** and **break_on_trigger** options, cannot be included in the command.

Also, note that this does not work the same when using a foreground monitor (unless the processor becomes halted) because the analyzer continues to capture states when the break to monitor occurs.

To break emulator execution on the analyzer trigger

- Enter a trigger state specification in the entry buffer; then, choose **Trace→Until ()**.
- Using the command line, use the **break_on_trigger** option to the **trace** command.

The **break_on_trigger** option to the **trace** command allows you to cause the emulator to break when the analyzer finds the trigger state.

Note that the actual break may be several cycles after the analyzer trigger.

Examples

To trace before source line 102 and cause the emulator to break into the monitor when the analyzer triggers:

```
trace before address main."main.c": line 102
break_on_trigger <RETURN>
```

Using the Sequencer

When you use the analyzer's sequencer, you can specify traces that trigger on a series, or sequence, of states. You can specify a state which, when found, causes the analyzer to restart the search for the sequence of states. Also, the analyzer's sequencer allows you to trace "windows" of code execution.

This section describes how to:

- Trigger on a sequence of states.
- Specify a global restart state.
- Trace "windows" of program execution.

The sequencing and windowing capabilities from within the Softkey Interface are not as powerful or flexible as they are from within the Terminal Interface. For example, in the Terminal Interface, you can specify different restart states for each sequence term and you can set up a windowing trace specification where the trigger does not have to be in the window. If you do not find the sequencing flexibility you need from within Softkey Interface, refer to the *68340 Emulator User's Guide for the Terminal Interface.*

To trigger after a sequence of states

- Use the **trace find_sequence** command.

The analyzer's sequencer has several levels (also called *sequence terms*). Each state in the series of states to be found before triggering, as well as the trigger state, is associated with a sequence term.

The sequencer works like this: The analyzer searches for the state associated with the first sequence term. When that state is captured, the analyzer starts searching for the state associated with the second term, and so on. The last sequence term used is associated with the trigger state. When the trigger state is captured the analyzer is triggered. Up to seven sequence terms and an optional occurrence count for each term are available.

Examples

In the demo program, suppose you wish to trigger on the following sequence of events: the "save_points" function, the "interrupt_sim" function, and finally the "do_sort" function. Also, suppose you wish to store only opcode fetches of the assembly language LINK A6,#0 instruction (data values that equal 4E56H) to show function entry addresses.

To set up the sequencing trace specification, enter the following trace command.

trace find_sequence `save_points` ***then*** `interrupt_sim`
trigger about `do_sort` ***only data*** `4e56h <RETURN>`

set source off `<RETURN>`

```
Trace List              Offset=0
Label:      Address                    Opcode or Status               time count
Base:       symbols                   mnemonic w/symbols                relative
-011    updat.write_hdwr   $4E56        supr prgm word rd (ds16)     ------------
sq adv  upda.save_points   $4E56        supr prgm word rd (ds16)       10.4   mS
sq adv  ma.interrupt_sim   $4E56        supr prgm word rd (ds16)        6.78  mS
-008    pr.proc_specific   $4E56        supr prgm word rd (ds16)       12.5   mS
-007    up.update_system  incomplete instr.: /4E56/????/                5.60  mS
-006    upda.get_targets  incomplete instr.: /4E56/????/               31.4   uS
-005    .read_conditions  incomplete instr.: /4E56/????/                3.50  mS
-004    upda.set_outputs  incomplete instr.: /4E56/????/                8.96  mS
-003    updat.write_hdwr  incomplete instr.: /4E56/????/               47.3   mS
-002    upda.save_points  incomplete instr.: /4E56/????/               10.4   mS
-001    ma.interrupt_sim  incomplete instr.: /4E56/????/                6.78  mS
about   pro|main.do_sort  incomplete instr.: /4E56/????/               16.4   mS
+001    pro|main.strcpy8  incomplete instr.: /4E56/????/                9.69  mS
+002    pro|main.strcpy8  incomplete instr.: /4E56/????/              450.    uS
+003    pro|main.strcpy8  incomplete instr.: /4E56/????/              450.    uS
+004    pro|main.strcpy8  incomplete instr.: /4E56/????/              450.    uS
```

Notice the states that contain "sq adv" in the first column (you may have to press <PREV> in order to see the states captured prior to the trigger). These are the states associated with (or captured for) each sequence term. Just as the trigger state is always stored in trace memory, the states captured in the sequence are always stored if the trace buffer is deep enough.

To specify a global restart state

- Use the **restart** option to the **trace** command.

When using the analyzer's sequencer, an additional sequence restart term is also allowed. This restart is a "global restart"; that is, it applies to all the sequence terms.

The restart term is a state which, when captured before the analyzer has found the trigger state, causes the search for the sequence of states to start over. You can use the restart term to make certain some state does not occur in the sequence that triggers the analyzer.

Examples

In the demo program, suppose you wish to trigger on the following sequence of events: the "save_points" function, the "interrupt_sim" function, and the "do_sort" function. However, you only want to trigger when the "interrupt_sim" calls the "do_sort" function. In other words, if the "proc_specific" function is entered before the "do_sort" function is entered, you know "interrupt_sim" did not call "do_sort" this time, and the analyzer should start searching again from the beginning.

Again, suppose you wish to store only opcode fetches of the assembly language LINK A6,#0 instruction (data values that equal 4E56H).

To set up this sequencing trace specification, enter the following trace command.

```
trace find_sequence save_points then interrupt_sim
restart proc_specific trigger about do_sort only data
4e56h <RETURN>
```

```
set source off <RETURN>
```

```
Trace List               Offset=0
Label:      Address                  Opcode or Status                   time count
Base:       symbols               mnemonic w/symbols                     relative
-012    upda.set_outputs  $4E56         supr prgm word rd (ds16)        109.     mS
-011    updat.write_hdwr  $4E56         supr prgm word rd (ds16)         47.3    mS
sq adv  upda.save_points  $4E56         supr prgm word rd (ds16)         10.4    mS
sq adv  ma.interrupt_sim  $4E56         supr prgm word rd (ds16)          6.78   mS
sq adv  pr.proc_specific  $4E56         supr prgm word rd (ds16)         86.0    uS
-007    up.update_system incomplete instr.: /4E56/????/                   5.60   mS
-006    upda.get_targets incomplete instr.: /4E56/????/                  31.5    uS
-005    .read_conditions incomplete instr.: /4E56/????/                   3.50   mS
-004    upda.set_outputs incomplete instr.: /4E56/????/                  51.9    mS
-003    updat.write_hdwr incomplete instr.: /4E56/????/                  47.3    mS
sq adv  upda.save_points incomplete instr.: /4E56/????/                  10.4    mS
sq adv  ma.interrupt_sim incomplete instr.: /4E56/????/                   6.78   mS
about   pro|main.do_sort incomplete instr.: /4E56/????/                 863.     uS
+001    pro|main.strcpy8 incomplete instr.: /4E56/????/                   9.69   mS
+002    pro|main.strcpy8 incomplete instr.: /4E56/????/                 450.     uS
+003    pro|main.strcpy8 incomplete instr.: /4E56/????/                 450.     uS
```

Notice in the preceding trace (you may have to press <PREV> in order to see the states captured prior to the trigger) that, in addition to states captured in the sequence, "sq adv" is also shown next to states which cause a sequencer restart.

To trace "windows" of program execution

- Use the **enable** and **disable** options to the **trace** command.

Windowing refers to the analyzer feature that allows you to turn on, or enable, the capturing of states after some state occurs then to turn off, or disable, the capturing of states when another state occurs. In effect, windowing allows you capture "windows" of code execution.

Windowing is different than storing states in a range (the **only range** option in the trace command syntax) because it allows you to capture execution of all states in a window of code whereas storing states in a range won’t capture the execution of subroutines that are called in that range or reads and writes to locations outside that range.

When you use the windowing feature of the analyzer, the trigger state must be in the window or else the trigger will never be found.

If you wish to combine the windowing and sequencing functions of the analyzer, there are some restrictions:

- Up to four sequence terms are available when windowing is in effect.
- Global restart is not available when windowing is in effect.
- Occurrence counts are not available.

Examples

In the demo program, suppose you are only interested in the execution that occurs within the switch statement of the "combsort" function. You could specify source line number 228 as the window enable state and the source line number of the next statement (line number 240) as the window disable state. Set up the windowing trace specification with the following command.

`trace enable` `main."main.c": line 229` ***`disable`***
`main."main.c": line 241 <RETURN>`

`set source on` `<RETURN>`

```
Trace List                 Offset=0                        More data off screen
Label:      Address            Opcode or Status w/ Source Lines        time count
Base:       symbols                   mnemonic w/symbols                relative
                }
+010    combsor+000000DE  NOP                                          680     nS
      ##########main.c - line   239 thru   241 ##################################

              /* Do the comb sort loop for this comb gap */
              for (top=len-gap,i=0; i < top; i++)
sq adv  combsor+000000E0  MOVEA.L  D3,A0                               720     nS
      ##########main.c - line   227 thru   229 ##################################

              /* Force gap to conform to comb11 */
              switch (gap)
sq adv  combsor+00000098  MOVE.L   (A3),D0                               6.06  mS
+013    combsor+0000009A  CMPI.L   #$0000000B,D0                       720     nS
+014    da|main.switches   $0000        supr data long wr (ds16)       720     nS
+015    switche+00000002   $0000        supr data word wr (ds16)       720     nS
+016     data|main.gap     $0000        supr data long rd (ds16)       720     nS
```

Notice in the resulting trace (you have to press the <NEXT> key) that the enable and disable states have the "sq adv" string in the line number column. This is because the windowing feature uses the analyzer’s sequencer.

Modifying the Trace Display

This section describes the options available when displaying trace lists.

This section describes how to:

- Display a dequeued trace.
- Display the trace about a line number.
- Display the trace, disassembling from a line number.
- Display instruction cycles only.
- Display the trace in absolute format.
- Display the trace in mnemonic format.
- Display the trace with high-level source lines.
- Display the trace with symbol information.
- Change the column widths in the trace display.
- Display time counts in absolute or relative format.
- Display the trace with address information offset by a value.
- Return to the default trace display.

To display a dequeued trace

- Use the **dequeue on** option to the **display trace** command.

Unused prefetches are removed from the trace display, and data transactions are aligned with the instructions that caused them to occur.

Examples

To display a dequeued trace:

display trace dequeue on <RETURN>

```
DQed Trace                  Offset=0
Label:  Address                      Opcode or Status                    time count
Base:     hex                            mnemonic                         relative
after   00000FC0  LINK.W   A6,#$0000                                   ------------
       =00012F94   stck sdata wr:$00012FF0
+004    00000FC4  JSR      $000014E8                                      2.9   uS
       =00012F90   stck sdata wr:$00000FCA
+010    000014E8  LINK.W   A6,#$0000                                      4.28  uS
       =00012F8C   stck sdata wr:$00012F94
+014    000014EC  MOVE.W   #$0049,$000072D6                               2.9   uS
       =000072D6   dest sdata wr:$0049
+018    000014F4  MOVE.W   #$002D,$000072D8                               2.8   uS
       =000072D8   dest sdata wr:$002D
+023    000014FC  MOVE.W   #$0044,$000076EE                               3.6   uS
       =000076EE   dest sdata wr:$0044
+028    00001504  MOVE.W   #$0029,$000076F0                               3.6   uS
       =000076F0   dest sdata wr:$0029
+033    0000150C  CLR.B    $00007700                                      3.6   uS
       =00007700   dest sdata wr:$00
```

To turn OFF the dequeued trace display:

display trace dequeue off <RETURN>

To display the trace about a line number

- Use the **<LINE #>** option to the **display trace** command.

The **<LINE #>** trace display option allows you to specify the line number to be centered in the display.

Examples

To display the trace about line number 66:

display trace `66 <RETURN>`

```
Trace List                    Offset=0
Label:  Address                     Opcode or Status                          time count
Base:     hex                           mnemonic                               relative
+059   00001532   $0000         supr prgm word rd (ds16)                      720      nS
+060   000076F6   $0000         supr data long wr (ds16)                      720      nS
+061   000076F8   $0000         supr data word wr (ds16)                      720      nS
+062   00001534   $154E         supr prgm word rd (ds16)                      720      nS
+063   00001536   $4239         supr prgm word rd (ds16)                      720      nS
+064   00012F88   $0000         supr data long wr (ds16)                      720      nS
+065   00012F8A   $1536         supr data word wr (ds16)                      720      nS
+066   0000154E   $4E56         supr prgm word rd (ds16)                      720      nS
+067   00001550   $0000         supr prgm word rd (ds16)                      680      nS
+068   00012F84   $0001         supr data long wr (ds16)                      720      nS
+069   00012F86   $2F8C         supr data word wr (ds16)                      720      nS
+070   00001552   $207C         supr prgm word rd (ds16)                      720      nS
+071   00001554   $0000         supr prgm word rd (ds16)                      720      nS
+072   00001556   $7156         supr prgm word rd (ds16)                      720      nS
+073   00001558   $7200         supr prgm word rd (ds16)                      720      nS
+074   0000155A   $4E71         supr prgm word rd (ds16)                      720      nS
```

To display the trace, disassembling from a line number

- Use the **disassemble_from_line_number** option to the **display trace** command.

The "disassemble_from_line_number" trace display option causes the inverse assembler to attempt to begin disassembling the trace information from the specified line number. This option is required for inverse assemblers that cannot uniquely identify opcode fetch states on the processor bus.

If the line number specified is not an opcode fetch state, the disassembled information will be incorrect.

Examples

To display the trace, disassembling from line number 66:

display trace disassemble_from_line_number 66 <RETURN>

```
Trace List               Offset=0
Label:  Address                    Opcode or Status                      time count
Base:     hex                          mnemonic                            relative
+066   0000154E  LINK.W   A6,#$0000                                        720     nS
+067   00001550   $0000         supr prgm word rd (ds16)                   680     nS
+068   00012F84   $0001         supr data long wr (ds16)                   720     nS
+069   00012F86   $2F8C         supr data word wr (ds16)                   720     nS
+070   00001552  MOVEA.L  #$00007156,A0                                    720     nS
+071   00001554   $0000         supr prgm word rd (ds16)                   720     nS
+072   00001556   $7156         supr prgm word rd (ds16)                   720     nS
+073   00001558  MOVEQ    #$00000000,D1                                    720     nS
+074   0000155A  NOP                                                      720     nS
+075   0000155C  MOVE.L   D1,D0                                            720     nS
+076   0000155E  ADD.L    D0,D0                                            720     nS
+077   00001560  ADD.L    D1,D0                                            680     nS
+078   00001562  LSL.L    #2,D0                                            720     nS
+079   00001564  MOVE.W   #$0041,($00,A0,D0.L)                             720     nS
+080   00001566   $0041         supr prgm word rd (ds16)                   720     nS
+081   00001568   $0800         supr prgm word rd (ds16)                   720     nS
```

To display instruction cycles only

- Use the **disassemble_from_line_number <LINE #> instructions_only** option to the **display trace** command.

Only the lines that contain an instruction opcode are displayed in the trace.

Examples

To display the trace about line number 66 displaying only instruction cycles:

display trace disassemble_from_line_number 66
instructions_only <RETURN>

```
Trace List                    Offset=0
Label:  Address                       Opcode or Status                      time count
Base:     hex                             mnemonic                            relative
+066   0000154E  LINK.W    A6,#$0000                                          37.2   uS
+070   00001552  MOVEA.L   #$00007156,A0                                       2.8   uS
+073   00001558  MOVEQ     #$00000000,D1                                       2.2   uS
+074   0000155A  NOP                                                          720    nS
+075   0000155C  MOVE.L    D1,D0                                              720    nS
+076   0000155E  ADD.L     D0,D0                                              720    nS
+077   00001560  ADD.L     D1,D0                                              680    nS
+078   00001562  LSL.L     #2,D0                                              720    nS
+079   00001564  MOVE.W    #$0041,($00,A0,D0.L)                               720    nS
+082   0000156A  MOVE.W    #$0029,($02,A0,D0.L)                                 2.2   uS
+086   00001570  CLR.L     ($04,A0,D0.L)                                        2.8   uS
+089   00001574  CLR.L     ($08,A0,D0.L)                                        2.2   uS
+093   00001578  NOP                                                          2.9    uS
+094   0000157A  ADDQ.L    #1,D1                                              720    nS
+097   0000157C  NOP                                                          2.1    uS
+098   0000157E  MOVEQ     #$00000020,D0                                      720    nS
```

To return to displaying all cycles:

display trace disassemble_from_line_number 66
all_cycles <RETURN>

To display the trace in absolute format

- Use the **absolute** option to the **display trace** command.

The **absolute** trace display option allows you to display status information in absolute format (binary, hex, or mnemonic). The **absolute status mnemonic** display is the same as default mnemonic display, except that opcodes are not disassembled.

Examples

To display the trace in absolute format with the status information as binary values:

display trace absolute status binary <RETURN>

```
Trace List                    Offset=0
Label:  Address    Data               Absolute Status                  time count
Base:     hex      hex                     binary                        relative
+066   0000154E    4E56  0000111111011101                                   720    nS
+067   00001550    0000  0010111111011101                                   680    nS
+068   00012F84    0001  0011111110001011                                   720    nS
+069   00012F86    2F8C  0011111111001011                                   720    nS
+070   00001552    207C  0010111111011101                                   720    nS
+071   00001554    0000  0010111111011101                                   720    nS
+072   00001556    7156  0010111111011101                                   720    nS
+073   00001558    7200  0010111111011101                                   720    nS
+074   0000155A    4E71  0010111111011101                                   720    nS
+075   0000155C    2001  0010111111011101                                   720    nS
+076   0000155E    D080  0010111111011101                                   720    nS
+077   00001560    D081  0010111111011101                                   680    nS
+078   00001562    E588  0010111111011101                                   720    nS
+079   00001564    31BC  0010111111011101                                   720    nS
+080   00001566    0041  0010111111011101                                   720    nS
+081   00001568    0800  0010111111011101                                   720    nS
```

To display the trace in mnemonic format

- Use the **mnemonic** option to the **display trace** command.

The **mnemonic** trace display option allows you to display the trace information in mnemonic format (that is, opcodes and status). The default trace display is in mnemonic format.

Examples

To display the trace in mnemonic format:

display trace mnemonic <RETURN>

```
Trace List                  Offset=0
Label:  Address                     Opcode or Status                     time count
Base:     hex                          mnemonic                           relative
+066    0000154E  LINK.W   A6,#$0000                                        720    nS
+067    00001550   $0000        supr prgm word rd (ds16)                    680    nS
+068    00012F84   $0001        supr data long wr (ds16)                    720    nS
+069    00012F86   $2F8C        supr data word wr (ds16)                    720    nS
+070    00001552  MOVEA.L  #$00007156,A0                                    720    nS
+071    00001554   $0000        supr prgm word rd (ds16)                    720    nS
+072    00001556   $7156        supr prgm word rd (ds16)                    720    nS
+073    00001558  MOVEQ    #$00000000,D1                                    720    nS
+074    0000155A  NOP                                                       720    nS
+075    0000155C  MOVE.L   D1,D0                                            720    nS
+076    0000155E  ADD.L    D0,D0                                            720    nS
+077    00001560  ADD.L    D1,D0                                            680    nS
+078    00001562  LSL.L    #2,D0                                            720    nS
+079    00001564  MOVE.W   #$0041,($00,A0,D0.L)                             720    nS
+080    00001566   $0041        supr prgm word rd (ds16)                    720    nS
+081    00001568   $0800        supr prgm word rd (ds16)                    720    nS
```

To display the trace with high-level source lines

- Use the **set source** command.

To include high-level source lines in the trace display, you must use the **set** command. The **set** command allows you to include symbolic information in trace, memory, register, and software breakpoint displays. The **set** command affects all displays for the current window.

The **set source on/off/only** command allows you to include source file information in the trace list or memory mnemonic display. The **source only** option specifies that only the source file information will be displayed.

When source lines are included, comments that contain file and line information appear before the source lines.

Also, when source lines are turned on, three additional options are available in the set command: inverse video, tabs are, and number of source lines.

The **inverse_video** option allows you to display source lines in inverse video.

The **tabs_are** option allows you to specify the number of spaces between tab stops so that the appropriate number of spaces can be inserted for source lines. The default value is eight. Values from two to 15 can be entered.

Typically, there are lines in the source file that are not associated with actual instructions (declarations, comments, etc.). The **number_of_source_lines** option allows you to specify the number of these source lines to be displayed for every source line that is associated with an actual instruction. Only source lines up to the the previous source line that corresponds to actual code will be displayed. The default value is five. Values from one to 50 can be entered.

Examples

To display the trace with high-level source lines:

```
set source on <RETURN>
display trace <RETURN>
```

```
Trace List               Offset=0
Label:  Address          Opcode or Status w/ Source Lines          time count
Base:     hex                            mnemonic                    relative
      ##########init_system.c - line    66 thru    80 ##########################
       * Returns: Nothing.
       ************************************************************************
      void
      init_val_arr()
      {
+066   0000154E  LINK.W   A6,#$0000                                    720     nS
+067   00001550   $0000         supr prgm word rd (ds16)               680     nS
+068   00012F84   $0001         supr data long wr (ds16)               720     nS
+069   00012F86   $2F8C         supr data word wr (ds16)               720     nS
+070   00001552  MOVEA.L  #$00007156,A0                                720     nS
+071   00001554   $0000         supr prgm word rd (ds16)               720     nS
+072   00001556   $7156         supr prgm word rd (ds16)               720     nS
      ##########init_system.c - line    81 thru    82 ##########################
          int cur_el;
          for (cur_el = 0; cur_el < NUM_OF_OLD; cur_el++)
```

To set the number of source lines to be displayed at 12:

set source on number_of_source_lines `12 <RETURN>`
display trace `<RETURN>`

```
Trace List               Offset=0
Label:  Address          Opcode or Status w/ Source Lines          time count
Base:     hex                            mnemonic                    relative
      ##########init_system.c - line    66 thru    80 ##########################
       *
       * Description: This code initializes the val_arr data structure.
       *
       * Parameters: none
       *
       * References: None.
       *
       * Returns: Nothing.
       ************************************************************************
      void
      init_val_arr()
      {
+066   0000154E  LINK.W   A6,#$0000                                    720     nS
+067   00001550   $0000         supr prgm word rd (ds16)               680     nS
+068   00012F84   $0001         supr data long wr (ds16)               720     nS
```

To display the trace with symbol information

The **set symbols on/off** command allows you to specify that address information be displayed in terms of program symbols.

Examples

To display the trace with symbol information:

set source off symbols on <RETURN>
display trace <RETURN>

```
Trace List              Offset=0
Label:      Address                        Opcode or Status                  time count
Base:       symbols                        mnemonic w/symbols                 relative
+066   ini.init_val_arr  LINK.W   A6,#$0000                                   720     nS
+067   init_va+00000002   $0000         supr prgm word rd (ds16)              680     nS
+068   sysstac+00007F84   $0001         supr data long wr (ds16)              720     nS
+069   sysstac+00007F86   $2F8C         supr data word wr (ds16)              720     nS
+070   init_va+00000004  MOVEA.L  #$00007156,A0                               720     nS
+071   init_va+00000006   $0000         supr prgm word rd (ds16)              720     nS
+072   init_va+00000008   $7156         supr prgm word rd (ds16)              720     nS
+073   init_va+0000000A  MOVEQ    #$00000000,D1                               720     nS
+074   init_va+0000000C  NOP                                                  720     nS
+075   init_va+0000000E  MOVE.L   D1,D0                                       720     nS
+076   init_va+00000010  ADD.L    D0,D0                                       720     nS
+077   init_va+00000012  ADD.L    D1,D0                                       680     nS
+078   init_va+00000014  LSL.L    #2,D0                                       720     nS
+079   init_va+00000016  MOVE.W   #$0041,($00,A0,D0.L)                        720     nS
+080   init_va+00000018   $0041         supr prgm word rd (ds16)              720     nS
+081   init_va+0000001A   $0800         supr prgm word rd (ds16)              720     nS
```

To change column widths in the trace display

- Use the **set width** command.

The **set width** command allows you to change the width of the address and mnemonic (or absolute) columns in the trace list. Values from one to 80 can be entered.

When address information is being displayed in terms of symbols (in other words, symbols on), you may wish to increase the width of the address column to display more of the symbol information.

When trace information is displayed in mnemonic format, you can additionally specify the width of symbols in the "Opcode or Status" column.

Examples

To display the trace with the address column width set to 30 characters:

set width label `30 <RETURN>`
display trace `<RETURN>`

```
Trace List                    Offset=0                           More data off screen
Label:            Address                                Opcode or Status
Base:         symbols                                    mnemonic w/symbols
+066   prog|init_system.init_val_arr   LINK.W   A6,#$0000
+067       prog|init_val_arr+00000002   $0000         supr prgm word rd (ds16)
+068          stack|sysstack+00007F84   $0001         supr data long wr (ds16)
+069          stack|sysstack+00007F86   $2F8C         supr data word wr (ds16)
+070       prog|init_val_arr+00000004  MOVEA.L  #$00007156,A0
+071       prog|init_val_arr+00000006   $0000         supr prgm word rd (ds16)
+072       prog|init_val_arr+00000008   $7156         supr prgm word rd (ds16)
+073       prog|init_val_arr+0000000A  MOVEQ    #$00000000,D1
+074       prog|init_val_arr+0000000C  NOP
+075       prog|init_val_arr+0000000E  MOVE.L   D1,D0
+076       prog|init_val_arr+00000010  ADD.L    D0,D0
+077       prog|init_val_arr+00000012  ADD.L    D1,D0
+078       prog|init_val_arr+00000014  LSL.L    #2,D0
+079       prog|init_val_arr+00000016  MOVE.W   #$0041,($00,A0,D0.L)
+080       prog|init_val_arr+00000018   $0041         supr prgm word rd (ds16)
+081       prog|init_val_arr+0000001A   $0800         supr prgm word rd (ds16)
```

To display time counts in absolute or relative format

- Use the **count** option to the **display trace** command.

Count information may be displayed two ways: relative (which is the default), or absolute. When relative is selected, count information is displayed relative to the previous state. When absolute is selected, count information is displayed relative to the trigger condition.

The **count absolute/relative** trace display option is not available when counting is turned off in the trace command.

Examples

To display the trace with absolute time counts:

display trace count absolute <RETURN>

```
Trace List              Offset=0
Label:   Address                   Opcode or Status                        time count
Base:      hex                         mnemonic                             absolute
+066    0000154E  LINK.W    A6,#$0000                                    +  47.28  uS
+067    00001550   $0000          supr prgm word rd (ds16)               +  47.96  uS
+068    00012F84   $0001          supr data long wr (ds16)               +  48.68  uS
+069    00012F86   $2F8C          supr data word wr (ds16)               +  49.40  uS
+070    00001552  MOVEA.L   #$00007156,A0                                +  50.12  uS
+071    00001554   $0000          supr prgm word rd (ds16)               +  50.84  uS
+072    00001556   $7156          supr prgm word rd (ds16)               +  51.56  uS
+073    00001558  MOVEQ     #$00000000,D1                                +  52.28  uS
+074    0000155A  NOP                                                    +  53.00  uS
+075    0000155C  MOVE.L    D1,D0                                        +  53.72  uS
+076    0000155E  ADD.L     D0,D0                                        +  54.44  uS
+077    00001560  ADD.L     D1,D0                                        +  55.12  uS
+078    00001562  LSL.L     #2,D0                                        +  55.84  uS
+079    00001564  MOVE.W    #$0041,($00,A0,D0.L)                         +  56.56  uS
+080    00001566   $0041          supr prgm word rd (ds16)               +  57.28  uS
+081    00001568   $0800          supr prgm word rd (ds16)               +  58.00  uS
```

To display the trace with addresses offset

- Use the **offset_by** option to the **display trace** command.

The **offset_by** trace display option allows you to cause the address information in the trace display to be offset by the amount specified. The offset value is subtracted from the instruction's physical address to yield the address that is displayed.

If code gets relocated and therefore makes symbolic information obsolete, you can use the **offset_by** option to change the address information so that it again agrees with the symbolic information.

You can also specify an offset to cause the listed addresses to match the addresses in compiler or assembler listings.

Examples

To display the trace with addresses offset by 154EH:

display trace offset_by `154eh <RETURN>`

```
Trace List               Offset=154E
Label:  Address                    Opcode or Status                   time count
Base:     hex                          mnemonic                        absolute
+066   00000000  LINK.W   A6,#$0000                                  +  47.28  uS
+067   00000002   $0000         supr prgm word rd (ds16)             +  47.96  uS
+068   00011A36   $0001         supr data long wr (ds16)             +  48.68  uS
+069   00011A38   $2F8C         supr data word wr (ds16)             +  49.40  uS
+070   00000004  MOVEA.L  #$00007156,A0                              +  50.12  uS
+071   00000006   $0000         supr prgm word rd (ds16)             +  50.84  uS
+072   00000008   $7156         supr prgm word rd (ds16)             +  51.56  uS
+073   0000000A  MOVEQ    #$00000000,D1                              +  52.28  uS
+074   0000000C  NOP                                                 +  53.00  uS
+075   0000000E  MOVE.L   D1,D0                                      +  53.72  uS
+076   00000010  ADD.L    D0,D0                                      +  54.44  uS
+077   00000012  ADD.L    D1,D0                                      +  55.12  uS
+078   00000014  LSL.L    #2,D0                                      +  55.84  uS
+079   00000016  MOVE.W   #$0041,($00,A0,D0.L)                       +  56.56  uS
+080   00000018   $0041         supr prgm word rd (ds16)             +  57.28  uS
+081   0000001A   $0800         supr prgm word rd (ds16)             +  58.00  uS
```

To return to the default trace display

- Use the **set default** command.

The **set default** command allows you to return to the default display.

Examples

To return to the default trace display:

set default <RETURN>

```
Trace List                    Offset=0
Label:  Address                      Opcode or Status                         time count
Base:     hex                           mnemonic                               relative
+066   0000154E  LINK.W   A6,#$0000                                            720      nS
+067   00001550   $0000         supr prgm word rd (ds16)                       680      nS
+068   00012F84   $0001         supr data long wr (ds16)                       720      nS
+069   00012F86   $2F8C         supr data word wr (ds16)                       720      nS
+070   00001552  MOVEA.L  #$00007156,A0                                        720      nS
+071   00001554   $0000         supr prgm word rd (ds16)                       720      nS
+072   00001556   $7156         supr prgm word rd (ds16)                       720      nS
+073   00001558  MOVEQ    #$00000000,D1                                        720      nS
+074   0000155A  NOP                                                           720      nS
+075   0000155C  MOVE.L   D1,D0                                                720      nS
+076   0000155E  ADD.L    D0,D0                                                720      nS
+077   00001560  ADD.L    D1,D0                                                680      nS
+078   00001562  LSL.L    #2,D0                                                720      nS
+079   00001564  MOVE.W   #$0041,($00,A0,D0.L)                                 720      nS
+080   00001566   $0041         supr prgm word rd (ds16)                       720      nS
+081   00001568   $0800         supr prgm word rd (ds16)                       720      nS
```

Saving and Restoring Traces

The emulator/analyzer interface allow you to save trace commands and trace lists. You can restore trace commands in order to set up the same trace specification. You can restore traces in order to view trace data captured in the stored trace.

This section describes how to:

- Save trace commands.
- Restore trace commands.
- Save traces.
- Restore traces.

To save trace commands

- Choose **File→Store→Trace Spec**.
- Using the command line, enter the **store trace_spec** command.

You can save a trace command to a "trace specification" file and reload it at a later time.

The trace command is saved in a file named "tspecfile.TS" in the current directory. The extension ".TS" is appended to trace specification files if no extension is specified in the **store trace_spec** command.

Examples

To store the current trace command:

`store trace_spec` `tspecfile <RETURN>`

To restore trace commands

- Choose **File**→**Load**→**Trace Spec**.
- Using the command line, enter the **load trace_spec** command.

Trace commands that are restored will always work, even if symbols have been changed; however, once you modify the trace command, it may no longer work.

Examples

To bring back the trace command saved in "tspecfile.TS" and perform a trace measurement using it:

load trace_spec `tspecfile <RETURN>`

trace again `<RETURN>`

To save traces

- Choose **File**→**Store**→**Trace Data**.
- Using the command line, enter the **store trace** command.

You can save a trace to a trace file and reload it at a later time.

The trace is saved in a file named "trcfile.TR" in the current directory. The extension ".TR" is appended to trace files if it is not specified in the **store trace** command.

Examples

To store the current trace:

store trace `trcfile <RETURN>`

To restore traces

- Choose **File→Load→Trace Data**.

- Using the command line, enter the **load trace** command.

The restored trace depth is the depth specified when the trace was stored and cannot be increased. You may want to increase the trace depth before storing traces.

When a trace is loaded, the trace command is not restored. A **trace again** or **trace modify** command will use the last trace command entered, not the command which resulted in the loaded trace. Also, the trace status shown by the **display status** command does not reflect the loaded trace.

Examples

To restore the "trcfile.TR" trace file:

load trace `trcfile <RETURN>`

The trace information stored in "trcfile.TR" is restored. You can view the trace as you would any other trace.

8

Making Software Performance Measurements

Making Software Performance Measurements

The Software Performance Measurement Tool (SPMT) is a feature of the Softkey Interface that allows you to make software performance measurements on your programs.

The SPMT allows you to make some of the measurements that are possible with the HP 64708 Software Performance Analyzer and its Graphical User Interface (HP B1487).

The SPMT post-processes information from the analyzer trace list. When you end a performance measurement, the SPMT dumps the post-processed information to a binary file, which is then read using the **perf32** report generator utility.

Two types of software performance measurements can be made with the SPMT: activity measurements, and duration measurements.

This chapter describes tasks you perform while using the Software Performance Measurement Tool (SPMT). These tasks are grouped into the following sections:

- Activity performance measurements.
- Duration performance measurements.
- Running performance measurements and creating reports.

Activity Performance Measurements

Activity measurements are measurements of the number of accesses (reads or writes) within an address range. The SPMT shows you the percentage of analyzer trace states that are in the specified address range, as well as the percentage of time taken by those states. Two types of activity are measured: memory activity, and program activity.

Memory activity is all activity that occurs within the address range.

Program activity is the activity caused by instruction execution in the address range. Program activity includes opcode fetches and the cycles that result from the execution of those instructions (reads and writes to memory, stack pushes, etc.).

For example, suppose an address range being measured for activity contains an opcode that causes a stack push, which results in multiple write operations to the stack area (outside the range). The memory activity measurement will count only the stack push opcode cycle. However, the program activity measurement will count the stack push opcode cycle and the write operations to the stack.

By comparing the program activity and the memory activity in an address range, you can get an idea of how much activity in other areas is caused by the code being measured. An activity measurement report of the code (prog), data, and stack sections of a program is shown below.

```
 Label

prog
      Address Range       ADEH thru     1261H


      Memory Activity
           State Percent  Rel =  57.77  Abs =  57.77
                         Mean = 295.80  Sdv =  26.77
           Time  Percent  Rel =  60.97  Abs =  60.97

      Program Activity
           State Percent  Rel =  99.82  Abs =  99.82
                         Mean = 511.10  Sdv =   0.88
           Time  Percent  Rel =  99.84  Abs =  99.84

data
      Address Range     6007AH thru    603A5H


      Memory Activity
           State Percent  Rel =  30.51  Abs =  30.51
                         Mean = 156.20  Sdv =  31.87
           Time  Percent  Rel =  28.09  Abs =  28.09
```

```
      Program Activity
           State Percent  Rel =   0.18  Abs =   0.18
                         Mean =   0.90  Sdv =   0.88
           Time  Percent  Rel =   0.16  Abs =   0.16

stack
      Address Range     40000H thru    43FFFH


      Memory Activity
           State Percent  Rel =  11.72  Abs =  11.72
                         Mean =  60.00  Sdv =  29.24
           Time  Percent  Rel =  10.94  Abs =  10.94

      Program Activity
           State Percent  Rel =   0.00  Abs =   0.00
                         Mean =   0.00  Sdv =   0.00
           Time  Percent  Rel =   0.00  Abs =   0.00



         Graph of Memory Activity relative state percents >= 1
prog                 57.77%  *****************************
data                 30.51%  ****************
stack                11.72%  ******

         Graph of Memory Activity relative time percents >= 1
prog                 60.97%  *******************************
data                 28.09%  **************
stack                10.94%  ******



         Graph of Program Activity relative state percents >= 1
prog                 99.82% **************************************************


         Graph of Program Activity relative time percents >= 1
prog                 99.84% **************************************************

      Summary Information for     10 traces


           Memory Activity
           State count
                Relative count     5120
                Mean sample      170.67
                Mean Standard Dv 29.30
                95% Confidence 12.28% Error tolerance
           Time  count
                Relative Time - Us 2221.20


           Program Activity
           State count
                Relative count     5120
                Mean sample      170.67
                Mean Standard Dv 0.58
                95% Confidence 0.24% Error tolerance
           Time  count
                Relative Time - Us 2221.20
      Absolute Totals
```

```
Absolute count - state     5120
Absolute count - time - Us 2221.20
```

This section describes how to:

- Set up the trace command for activity measurements.
- Initialize activity performance measurements.
- Interpret activity measurement reports.

To set up the trace command for activity measurements

1 Specify a trace display depth of 512.

2 Trace after any state, store all states, and count time.

Before you initialize and run performance measurements, the current trace command (in other words, the last trace command entered) must be properly set up.

In general, you want to give the SPMT as many trace states as possible to post-process, so you should increase the trace depth to the maximum number, as shown in the following command.

If you wish to measure activity as a percentage of all activity, the current trace command should be the default (in other words, **trace <RETURN>**). The default trace command triggers on any state, and all captured states are stored. It is important that time be counted by the analyzer; otherwise, the SPMT measurements will not be correct. Also, since states are stored "after" the trigger state, the maximum number of captured states appears in each trace list.

You can qualify trace commands any way you like to obtain specific information. However, when you qualify the states that get stored in the trace memory, your SPMT results will be biased by your qualifications; the percentages shown will be of only those states stored in the trace list.

Examples

To specify a trace depth of 512:

display trace depth `512 <RETURN>`

To trace after any state, store all states, and count time:

trace counting time `<RETURN>`

To initialize activity performance measurements

- Use the **performance_measurement_initialize** command.

After you set up the trace command, you must tell the SPMT the address ranges on which you wish to make activity measurements. This is done by initializing the performance measurement. You can initialize the performance measurement in the following ways:

- Default initialization (using global symbols if the symbols database is loaded).
- Initialize with user-defined files.
- Initialize with global symbols.
- Initialize with local symbols.
- Restore a previous performance measurement (if the emulation system has been exited and reentered).

Default Initialization

Entering the **performance_measurement_initialize** command with no options specifies an activity measurement. If a valid symbolic database has been loaded, the addresses of all global procedures and static symbols will be used; otherwise, a default set of ranges that cover the entire processor address range will be used.

Initialization with User Defined Ranges

You can specifically give the SPMT address ranges to use by placing the information in a file and entering the file name in the **performance_measurement_initialize** command.

Address range files may contain program symbols (procedure name or static), user defined address ranges, and comments. An example address range file is shown below.

```
# Any line which starts with a # is a comment.
# All user's labels must be preceded by a "|".

|users_label  10H 1000H
program_symbol

# A program symbol can be a procedure name or a static.  In the case of a pro-
# cedure name the range of that procedure will be used.

|users_label2 program_symbol1 -> program_symbol2

# "->" means thru.  The above will define a range which starts with symbol1
# and goes thru symbol2.  If both symbols are procedures then the range will
# be defined as the start of symbol1 thru the end of symbol2.

dir1/dir2/source_file.s:local_symbol

# The above defines a range based on the address of local_symbol.
```

Initialization with Global Symbols

When the **performance_measurement_initialize** command is entered with no options or with the **global_symbols** option, the global symbols in the symbols database become the address ranges for which activity is measured. If the symbols database is not loaded, a default set of ranges that cover the entire processor address range will be used.

The global symbols database contains procedure symbols, which are associated with the address range from the beginning of the procedure to the end, and static symbols, which are associated with the address of the static variable.

Initialization with Local Symbols

When the **performance_measurement_initialize** command is entered with the **local_symbols_in** option and a source file name, the symbols associated with that source file become the address ranges for which activity is measured. If the symbols database is not loaded, an error message will occur telling you that the source filename symbol was not found.

You can also use the **local_symbols_in** option with procedure symbols; this allows you to measure activity related to the symbols defined in a single function or procedure.

Restoring the Current Measurement

The **performance_measurement_initialize restore** command allows you to restore old performance measurement data from the **perf.out** file in the current directory.

If you have not exited and reentered emulation, you can add traces to a performance measurement simply by entering another **performance_measurement_run** command. However, if you exit and reenter the emulation system, you must enter the **performance_measurement _initialize restore** command before you can add traces to a performance measurement. When you restore a performance measurement, make sure your current trace command is identical to the command used with the restored measurement.

The **restore** option checks the emulator software version and will only work if the **perf.out** files you are restoring were made with the same software version as is presently running in the emulator. If you ran tests using a former software version and saved **perf.out** files, then updated your software to a new version number, you will not be able to restore old **perf.out** measurement files.

Examples

Suppose the "addr_ranges" file contains the names of all the functions in the "ecs" demo program loop:

```
combsort
do_sort
gen_ascii_data
get_targets
graph_data
interrupt_sim
proc_specific
read_conditions
save_points
set_outputs
strcpy8
update_system
write_hdwr
```

Since these labels are program symbols, you do not have to specify the address range associated with each label; the SPMT will search the symbol database for the addresses of each label.

An easy way to create the "addr_ranges" file is to use the **copy global_symbols** command to copy the global symbols to a file named "addr_ranges"; then, fork a shell to UNIX (by entering "! <RETURN>" on the Softkey Interface command line) and edit the file so that it contains the procedure names shown above. Enter a **<CTRL>d** at the UNIX prompt to return to the Softkey Interface.

To initialize the activity measurement with a user-defined address range file:

performance_measurement_initialize `addr_ranges <RETURN>`

To interpret activity measurement reports

- View the performance measurement report.

Activity measurements are measurements of the number of accesses (reads or writes) within an address range. The reports generated for activity measurements show you the percentage of analyzer trace states that are in the specified address range, as well as the percentage of time taken by those states. The performance measurement must include four traces before statistics (mean and standard deviation) appear in the activity report. The information you will see in activity measurement reports is described below.

Memory Activity All activity found within the address range.

Program Activity All activity caused by instruction execution in the address range. Program activity includes opcode fetches and the cycles that result from the execution of those instructions (reads and writes to memory, stack pushes, etc.).

Relative With respect to activity in all ranges defined in the performance measurement.

Absolute With respect to all activity, not just activity in those ranges defined in the performance measurement.

Mean Average number of states in the range per trace. The following equation is used to calculate the mean:

$$mean = \frac{states\ \ in\ \ range}{total\ \ states}$$

Standard Deviation Deviation from the mean of state count. The following equation is used to calculate standard deviation:

$$std\ dev = \sqrt{\frac{1}{N-1} \times \sum_{i=1}^{N} S_{sumq} - N\ (mean)^2}$$

Where:

N	Number of traces in the measurement.
mean	Average number of states in the range per trace.
S_{sumq}	Sum of squares of states in the range per trace.

Symbols Within Range Names of other symbols that identify addresses or ranges of addresses within the range of this symbol.

Additional Symbols for Address Names of other symbols that also identify this address.

Note that some compilers emit more than one symbol for certain addresses. For example, a compiler may emit "interrupt_sim" and "_interrupt_sim" for the first address in a routine named interrupt_sim. The analyzer will show the first symbol it finds to represent a range of addresses, or a single address point, and it will show the other symbols under either "Symbols within range" or "Additional symbols for address", as applicable. In the "interrupt_sim" example, it may show either "interrupt_sim" or "_interrupt_sim" to represent the range, depending on which symbol it finds first. The other symbol will be shown below "Symbols within range" in the report. These conditions appear particularly in default measurements that include all global and local symbols.

Relative and Absolute Counts Relative count is the total number of states associated with the address ranges in the performance measurement. Relative time is the total amount of time associated with the address ranges in the performance measurement. The absolute counts are the number of states or amount of time associated with all the states in all the traces.

Error Tolerance and Confidence Level An approximate error may exist in displayed information. Error tolerance for a level of confidence is calculated using the mean of the standard deviations and the mean of the means. Error tolerance gives an indication of the stability of the information. For example, if the error is 5% for a confidence level of 95%, then you can be 95% confident that the information has an error of 5% or less.

The Student's "T" distribution is used in these calculations because it improves the accuracy for small samples. As the size of the sample increases, the Student's "T" distribution approaches the normal distribution.

The following equation is used to calculate error tolerance:

$$\textit{error pct.} = \frac{O_{\mathrm{m}} \times t}{N \times P_{\mathrm{m}}} \times 100$$

Where:

O_m	Mean of the standard deviations.
t	Table entry in Student's "T" table for a given confidence level.
N	Number of traces in the measurement.
P_m	Mean of the means (i.e., mean sample).

Examples

Consider the following activity measurement report (generated with the commands shown):

```
display trace depth 512 <RETURN>
trace counting time <RETURN>
performance_measurement_initialize addr_ranges <RETURN>
performance_measurement_run 20 <RETURN>
performance_measurement_end <RETURN>
!perf32 | more
```

```
 Label

set_outputs
         Address Range      177AH thru     180AH

         Memory Activity
                 State Percent  Rel =  31.31  Abs =  25.00
                               Mean = 128.00  Sdv = 227.46
                 Time  Percent  Rel =  31.30  Abs =  24.95

         Program Activity
                 State Percent  Rel =  28.61  Abs =  25.00
                               Mean = 128.00  Sdv = 227.46
                 Time  Percent  Rel =  28.57  Abs =  24.95

strcpy8
         Address Range      10A6H thru     1100H

         Memory Activity
                 State Percent  Rel =  21.77  Abs =  17.38
                               Mean =  89.00  Sdv = 149.70
                 Time  Percent  Rel =  21.76  Abs =  17.34

         Program Activity
                 State Percent  Rel =  26.39  Abs =  23.06
                               Mean = 118.05  Sdv = 190.05
                 Time  Percent  Rel =  26.47  Abs =  23.11

interrupt_sim
         Address Range      1014H thru     109EH

         Memory Activity
                 State Percent  Rel =  18.78  Abs =  15.00
                               Mean =  76.80  Sdv = 187.57
                 Time  Percent  Rel =  18.78  Abs =  14.97

         Program Activity
                 State Percent  Rel =  17.17  Abs =  15.00
                               Mean =  76.80  Sdv = 187.57
                 Time  Percent  Rel =  17.14  Abs =  14.97

write_hdwr
         Address Range      1812H thru     188AH

         Memory Activity
                 State Percent  Rel =  12.52  Abs =  10.00
                               Mean =  51.20  Sdv = 157.59
                 Time  Percent  Rel =  12.52  Abs =   9.98

         Program Activity
                 State Percent  Rel =  11.44  Abs =  10.00
                               Mean =  51.20  Sdv = 157.59
                 Time  Percent  Rel =  11.43  Abs =   9.98

get_targets
         Address Range      1654H thru     16DCH

         Memory Activity
                 State Percent  Rel =   6.26  Abs =   5.00
```

```
                             Mean =  25.60  Sdv = 114.49
              Time  Percent  Rel =   6.26  Abs =   4.99

      Program Activity
              State Percent  Rel =   5.72  Abs =   5.00
                             Mean =  25.60  Sdv = 114.49
              Time  Percent  Rel =   5.71  Abs =   4.99

update_system
      Address Range      1592H thru     164CH

      Memory Activity
              State Percent  Rel =   6.26  Abs =   5.00
                             Mean =  25.60  Sdv = 114.49
              Time  Percent  Rel =   6.26  Abs =   4.99

      Program Activity
              State Percent  Rel =   5.73  Abs =   5.01
                             Mean =  25.65  Sdv = 114.48
              Time  Percent  Rel =   5.73  Abs =   5.00

combsort
      Address Range      1244H thru     143AH

      Memory Activity
              State Percent  Rel =   2.08  Abs =   1.66
                             Mean =   8.50  Sdv =  21.92
              Time  Percent  Rel =   2.08  Abs =   1.66

      Program Activity
              State Percent  Rel =   3.59  Abs =   3.13
                             Mean =  16.05  Sdv =  41.49
              Time  Percent  Rel =   3.58  Abs =   3.12

do_sort
      Address Range      1442H thru     14E0H

      Memory Activity
              State Percent  Rel =   0.62  Abs =   0.50
                             Mean =   2.55  Sdv =   6.23
              Time  Percent  Rel =   0.63  Abs =   0.50

      Program Activity
              State Percent  Rel =   0.77  Abs =   0.67
                             Mean =   3.45  Sdv =   8.43
              Time  Percent  Rel =   0.78  Abs =   0.68

gen_ascii_data
      Address Range      1108H thru     123CH

      Memory Activity
              State Percent  Rel =   0.39  Abs =   0.31
                             Mean =   1.60  Sdv =   5.18
              Time  Percent  Rel =   0.40  Abs =   0.32

      Program Activity
              State Percent  Rel =   0.58  Abs =   0.51
                             Mean =   2.60  Sdv =   8.52
              Time  Percent  Rel =   0.59  Abs =   0.51
```

```
graph_data
         Address Range      197EH thru     1A36H

         Memory Activity
                 State Percent  Rel =   0.00  Abs =   0.00
                               Mean =   0.00  Sdv =   0.00
                 Time  Percent  Rel =   0.00  Abs =   0.00

         Program Activity
                 State Percent  Rel =   0.00  Abs =   0.00
                               Mean =   0.00  Sdv =   0.00
                 Time  Percent  Rel =   0.00  Abs =   0.00

proc_specific
         Address Range      1A62H thru     1A82H

         Memory Activity
                 State Percent  Rel =   0.00  Abs =   0.00
                               Mean =   0.00  Sdv =   0.00
                 Time  Percent  Rel =   0.00  Abs =   0.00

         Program Activity
                 State Percent  Rel =   0.00  Abs =   0.00
                               Mean =   0.00  Sdv =   0.00
                 Time  Percent  Rel =   0.00  Abs =   0.00

read_conditions
         Address Range      16E4H thru     1772H

         Memory Activity
                 State Percent  Rel =   0.00  Abs =   0.00
                               Mean =   0.00  Sdv =   0.00
                 Time  Percent  Rel =   0.00  Abs =   0.00

         Program Activity
                 State Percent  Rel =   0.00  Abs =   0.00
                               Mean =   0.00  Sdv =   0.00
                 Time  Percent  Rel =   0.00  Abs =   0.00

save_points
         Address Range      1892H thru     1976H

         Memory Activity
                 State Percent  Rel =   0.00  Abs =   0.00
                               Mean =   0.00  Sdv =   0.00
                 Time  Percent  Rel =   0.00  Abs =   0.00

         Program Activity
                 State Percent  Rel =   0.00  Abs =   0.00
                               Mean =   0.00  Sdv =   0.00
                 Time  Percent  Rel =   0.00  Abs =   0.00
```

```
         Graph of Memory Activity relative state percents >= 1
set_outputs           31.31%  ****************
strcpy8               21.77%  ***********
interrupt_sim         18.78%  **********
write_hdwr            12.52%  *******
get_targets            6.26%  ***
update_system          6.26%  ***
combsort               2.08%  *


         Graph of Memory Activity relative time percents >= 1
set_outputs           31.30%  ****************
strcpy8               21.76%  ***********
interrupt_sim         18.78%  **********
write_hdwr            12.52%  *******
get_targets            6.26%  ***
update_system          6.26%  ***
combsort               2.08%  *


         Graph of Program Activity relative state percents >= 1
set_outputs           28.61%  ***************
strcpy8               26.39%  *************
interrupt_sim         17.17%  *********
write_hdwr            11.44%  ******
get_targets            5.72%  ***
update_system          5.73%  ***
combsort               3.59%  **


         Graph of Program Activity relative time percents >= 1
set_outputs           28.57%  ***************
strcpy8               26.47%  *************
interrupt_sim         17.14%  *********
write_hdwr            11.43%  ******
get_targets            5.71%  ***
update_system          5.73%  ***
combsort               3.58%  **
```

```
Summary Information for     20 traces

        Memory Activity
        State count
                Relative count     8177
                Mean sample       31.45
                Mean Standard Dv 75.74
                95% Confidence 112.76% Error tolerance
        Time  count
                Relative Time - Us 5838.56

        Program Activity
        State count
                Relative count     8948
                Mean sample       34.42
                Mean Standard Dv 80.77
                95% Confidence 109.90% Error tolerance
        Time  count
                Relative Time - Us 6396.40
Absolute Totals
                Absolute count - state    10240
                Absolute count - time - Us 7325.88
```

The measurements for each label are printed in descending order according to the amount of activity. You can see that the set_outputs function has the most activity. Also, you can see that no activity is recorded for several of the functions. The histogram portion of the report compares the activity in the functions that account for at least 1% of the activity for all labels defined in the measurement.

Duration Performance Measurements

Duration measurements provide a best-case/worst-case characterization of code execution time. These measurements record execution times that fall within a set of specified time ranges. The analyzer trace command is set up to store only the entry and exit states of the module to be measured (for example, a C function or Pascal procedure). The SPMT provides two types of duration measurements: module duration, and module usage.

Module duration measurements record how much time it takes to execute a particular code segment (for example, a function in the source file).

Module usage shows how much of the execution time is spent outside of the module (from exit to entry). This measurement gives an indication of how often the module is being used.

When using the SPMT to perform duration measurements, there should be only two addresses stored in the trace memory: the entry address, and the exit address. Recursion can place several entry addresses before the first exit address, and/or several exit addresses before the first entry address. Duration measurements are made between the last entry address in a series of entry addresses, and the last exit address in a series of exit addresses (see the figure below). All of the entry and exit addresses which precede these last addresses are assumed to be unused prefetches, and are ignored during time measurements.

```
START  - assumed prefetch
START  - assumed prefetch
START  - assumed prefetch
START  - last ENTRY address -
END    - assumed prefetch
END    - assumed prefetch            Measure duration
END    - assumed prefetch
END    - last EXIT address  -
START  - assumed prefetch
START  - assumed prefetch            Measure duration
START  - assumed prefetch
START  - last ENTRY address -
END    - assumed prefetch
END    - assumed prefetch
```

When measuring a recursive function, module duration will be measured between the last recursive call and the true end of the recursive execution. This will affect the accuracy of the measurement.

If a module is entered at the normal point, and then exited by a point other than the defined exit point, the entry point will be ignored. It will be judged the same as any other unused prefetch, and no time-duration measurement will be made. Its time will be included in the measure of time spent outside the procedure or function.

If a module is exited from the normal point, and then reentered from some other point, the exit will also be assumed to be an unused prefetch of the exit state.

Note that if you are making duration measurements on a function that is recursive, or one that has multiple entry and/or exit points, you may wind up with invalid information.

This section describes how to:

- Set up the trace command for duration measurements.
- Initialize duration performance measurements.
- Interpret duration measurement reports.

To set up the trace command for duration measurements

1 Specify a trace display depth of 512.

2 Trace after and store only function start and end addresses.

For duration measurements, the trace command must be set up to store only the entry and exit points of the module of interest. Since the trigger state is always stored, you should trigger on the entry or exit points. For example:

trace after `symbol_entry` ***or*** `symbol_exit` ***only*** `symbol_entry` ***or*** `symbol_exit` ***counting time*** <RETURN>

CAUTION The previous command depends on the generation of correct exit address symbols by the software development tools.

Or:

trace after module_name start ***or*** module_name end ***only*** module_name start ***or*** module_name end ***counting time*** <RETURN>

Where "symbol_entry" and "symbol_exit" are symbols from the user program. Or, where "module_name" is the name of a C function or Pascal procedure (and is listed as a procedure symbol in the global symbol display).

Examples

To specify a trace display depth of 512:

display trace depth 512 <RETURN>

To set up the trace command for duration measurements on the interrupt_sim function:

trace after interrupt_sim start ***or*** interrupt_sim end ***only*** interrupt_sim start ***or*** interrupt_sim end ***counting time*** <RETURN>

The trace specification sets up the analyzer to capture only the states that contain the start address of the interrupt_sim function or the end address of the interrupt_sim function. Since the trigger state is also stored, the analyzer is set up to trigger on the entry or exit address of the interrupt_sim function. With these states in memory, the analyzer will derive two measurements: time from start to end of interrupt_sim, and time from end to start of interrupt_sim.

To initialize duration performance measurements

- Use the **performance_measurement_initialize** command with the **duration** option.

After you set up the trace command, you must tell the SPMT the time ranges to be used in the duration measurement. This is done by initializing the performance measurement. You can initialize the performance measurement in the following ways:

- Initialize with user-defined files.
- Restore a previous performance measurement (if the emulation system has been exited and reentered).

Initialization with User Defined Ranges

You can specifically give the SPMT time ranges to use by placing the information in a file and entering the file name in the **performance_measurement_initialize** command.

Time range files may contain comments and time ranges in units of microseconds (us), milliseconds (ms), or seconds (s). An example time range file is shown below.

```
# Any line which starts with a # is a comment.

1 us 20 us
10.1 ms 100.6 ms
3.55 s  6.77 s

# us microseconds
# ms milliseconds
#  s seconds
#
# The above are the only abbreviations allowed.  The space between the  number
# and the units abbreviation is required.
```

When no user defined time range file is specified, the following set of default time ranges are used.

```
1 us 10 us
10.1 us 100 us
100.1 us 500 us
500.1 us 1 ms
1.001 ms 5 ms
5.001 ms 10 ms
10.1 ms 20 ms
20.1 ms 40 ms
40.1 ms 80 ms
80.1 ms 160 ms
160.1 ms 320 ms
320.1 ms 640 ms
640.1 ms 1.2 s
```

Restoring the Current Measurement

The **performance_measurement_initialize restore** command allows you to restore old performance measurement data from the **perf.out** file in the current directory.

If you have not exited and reentered emulation, you can add traces to a performance measurement simply by entering another **performance_measurement_run** command. However, if you exit and reenter the emulation system, you must enter the **performance_measurement _initialize restore** command before you can add traces to a performance measurement. When you restore a performance measurement, make sure your current trace command is identical to the command used with the restored measurement.

The **restore** option checks the emulator software version and will only work if the **perf.out** files you are restoring were made with the same software version as is presently running in the emulator. If you ran tests using a former software version and saved **perf.out** files, then updated your software to a new version number, you will not be able to restore old **perf.out** measurement files.

Examples

To initialize the duration measurement:

performance_measurement_initialize duration <RETURN>

To interpret duration measurement reports

- View the performance measurement report.

Duration measurements provide a best-case/worst-case characterization of code execution time. These measurements record execution times that fall within a set of specified time ranges. The information you will see in duration measurement reports is described below.

Number of Intervals Number of "from address" and "to address" pairs (after prefetch correction).

Maximum Time The greatest amount of time between the "from address" to the "to address".

Minimum Time The shortest amount of time between the "from address" to the "to address".

Average Time Average time between the "from address" and the "to address". The following equation is used to calculate the average time:

$$mean = \frac{amount\ \ of\ \ time\ \ \text{for}\ \ all\ \ intervals}{number\ \ of\ \ intervals}$$

Standard Deviation Deviation from the mean of time. The following equation is used to calculate standard deviation:

$$std\ dev = \sqrt{\frac{1}{N-1} \times \sum_{i=1}^{N} S_{\text{sumq}} - N\ (mean)^2}$$

Where:

N	Number of intervals.
mean	Average time.
S_{sumq}	Sum of squares of time in the intervals.

Error Tolerance and Confidence Level An approximate error may exist in displayed information. Error tolerance for a level of confidence is calculated using the mean of the standard deviations and the mean of the means. Error tolerance gives an indication of the stability of the information. For example, if the error is 5% for a confidence level of 95%, then you can be 95% confident that the information has an error of 5% or less.

The Student's "T" distribution is used in these calculations because it improves the accuracy for small samples. As the size of the sample increases, the Student's "T" distribution approaches the normal distribution.

The following equation is used to calculate error tolerance:

$$error\ pct. = \frac{O_{\text{m}} \times t}{N \times P_{\text{m}}} \times 100$$

Where:

O_m	Mean of the standard deviations in each time range.
t	Table entry in Student's "T" table for a given confidence level.
N	Number of intervals.

P_m — Mean of the means (i.e., mean of the average times in each time range).

Examples

Consider the following duration measurement report (generated with the commands shown):

display trace depth 512 <RETURN>
trace after interrupt_sim start ***or*** interrupt_sim end ***only*** interrupt_sim start ***or*** interrupt_sim end ***counting time*** <RETURN>
performance_measurement_initialize duration <RETURN>
performance_measurement_run 10 <RETURN>
performance_measurement_end <RETURN>
!perf32 | more

```
                  Time Interval Profile


 From Address      1014
         File main(module)."/users/guest/demo/debug_env/hp64751/main.c"
         Symbolic Reference at main.interrupt_sim
 To Address        109E
         File main(module)."/users/guest/demo/debug_env/hp64751/main.c"
         Symbolic Reference at interrupt_sim+8A
 Number of intervals   2550
 Maximum Time 648396.800 us
 Minimum Time 65.640 us
 Avg Time     67133.955 us


         Statistical summary - for     10 traces
                 Stdv 139265.45
                 95% Confidence 8.05% Error tolerance


         Graph of relative percents
 1 us 10 us             0.00%
 10.1 us 100 us        15.06%  ********
 100.1 us 500 us        0.00%
 500.1 us 1 ms          4.98%  ***
 1.001 ms 5 ms         15.02%  ********
 5.001 ms 10 ms        14.82%  ********
 10.1 ms  20 ms        14.94%  ********
 20.1 ms  40 ms        10.00%  *****
 40.1 ms  80 ms         4.94%  ***
 80.1 ms  160 ms        9.57%  *****
 160.1 ms 320 ms        4.86%  ***
 320.1 ms 640 ms        5.53%  ***
 640.1 ms 1.2 s         0.27%
```

```
From Address      109E
         File main(module)."/users/guest/demo/debug_env/hp64751/main.c"
         Symbolic Reference at interrupt_sim+8A
To Address        1014
         File main(module)."/users/guest/demo/debug_env/hp64751/main.c"
         Symbolic Reference at main.interrupt_sim
Number of intervals   2550
Maximum Time 182927.360 us
Minimum Time 82534.400 us
Avg Time     99593.389 us


         Statistical summary - for     10 traces
                 Stdv 25497.98
                 95% Confidence 0.99% Error tolerance


         Graph of relative percents
1 us 10 us              0.00%
10.1 us 100 us          0.00%
100.1 us 500 us         0.00%
500.1 us 1 ms           0.00%
1.001 ms 5 ms           0.00%
5.001 ms 10 ms          0.00%
10.1 ms  20 ms          0.00%
20.1 ms  40 ms          0.00%
40.1 ms  80 ms          0.00%
80.1 ms  160 ms        96.94%  ************************************************
160.1 ms 320 ms         3.06%  **
320.1 ms 640 ms         0.00%
640.1 ms 1.2 s          0.00%
```

Two sets of information are given in the duration measurement report: module duration and module usage.

The first set of information in the duration measurement report is the "module duration" measurement. The module duration report shows that the amount of time it takes for the interrupt_sim function to execute varies from 65.64 microseconds to 648.4 milliseconds. The average amount of time it takes for the interrupt_sim module to execute is roughly 67.13 milliseconds.

The second set is the "module usage" measurement. Module usage measurements show how much time is spent outside the module of interest; they indicate how often the module is used. The information shown in the first part of the duration report above shows that the average amount of time spent outside the interrupt_sim function is about 99.59 milliseconds.

Running Measurements and Creating Reports

Several performance measurement tasks are the same whether you are making activity or duration measurements.

This section describes how to:

- Run performance measurements.
- End performance measurements.
- Create a performance measurement report.

To run performance measurements

- Use the **performance_measurement_run** command.

The **performance_measurement_run** command processes analyzer trace data. When you end the performance measurement, this processed data is dumped to the binary "perf.out" file in the current directory. The **perf32** report generator utility is used to read the binary information in the "perf.out" file.

If the **performance_measurement_run** command is entered without a count, the current trace data is processed. If a count is specified, the current trace command is executed consecutively the number of times specified. The data that results from each trace command is processed and combined with the existing processed data. The STATUS line will say "Processing trace <NO.>" during the run so you will know how your measurement is progressing. The only way to stop this series of traces is by using **<CTRL>c** (sig INT).

The more traces you include in your sample, the more accurate will be your results. At least four consecutive traces are required to obtain statistical interpretation of activity measurement results.

Examples

To run the performance measurement, enter the following command:

performance_measurement_run 20 <RETURN>

The command above causes 20 traces to occur. The SPMT processes the trace information after each trace, and the number of the trace being processed is shown on the status line.

To end performance measurements

- Use the **performance_measurement_end** command.

The **performance_measurement_end** command takes the data generated by the **performance_measurement_run** command and places it in a file named **perf.out** in the current directory. If a file named "perf.out" already exists in the current directory, it will be overwritten. Therefore, if you wish to save a performance measurement, you must rename the **perf.out** file before performing another measurement.

The **performance_measurement_end** command does not affect the current performance measurement data which exists within the emulation system. In other words, you can add more traces later to the existing performance measurement by entering another **performance_measurement_run** command.

Once you have entered the **performance_measurement_end** command, you can use the **perf32** report generator to look at the data saved in the **perf.out** file.

Note that the "perf.out" file is a binary file. Do not try to read it with the UNIX **more** or **cat** commands. The **perf32** report generator utility (described in the following section) must be used to read the contents of the "perf.out" file.

Examples

To cause the processed trace information to be dumped to the "perf.out" file:

performance_measurement_end <RETURN>

To create a performance measurement report

- Use the **perf32** command at the UNIX prompt.

The **perf32** report generator utility must be used to read the information in the "perf.out" file and other files dumped by the SPMT (in other words, renamed "perf.out" files). The **perf32** utility is run from the UNIX shell. You can fork a shell while in the Softkey Interface and run **perf32**, or you can exit the Softkey Interface and run **perf32**.

Options to "perf32"

A default report, containing all performance measurement information, is generated when the **perf32** command is used without any options. The options available with **perf32** allow you to limit the information in the generated report. These options are described below.

-h	Produce outputs limited to histograms.
-s	Produce a summary limited to the statistical data.
-p	Produce a summary limited to the program activity.
-m	Produce a summary limited to the memory activity.
-f<file>	Produce a report based on the information contained in <file> instead of the information contained in perf.out.

For example, the following commands save the current performance measurement information in a file called "perf1.out", and produce a histogram showing only the program activity occupied by the functions and variables.

```
mv perf.out perf1.out <RETURN>
perf32 -hpf perf1.out <RETURN>
```

Options **-h**, **-s**, **-p**, and **-m** affect the contents of reports generated for activity measurements. These options have no effect on the contents of reports generated for duration (time interval) measurements.

Examples

Now, to generate a report from the "perf.out" file, type the following on the command line to fork a shell and run the **perf32** utility:

```
!perf32 | more
```

9

Making Coordinated Measurements

Making Coordinated Measurements

When HP 64700 Card Cages are connected together via the Coordinated Measurement Bus (CMB), you can start and stop up to 32 emulators at the same time.

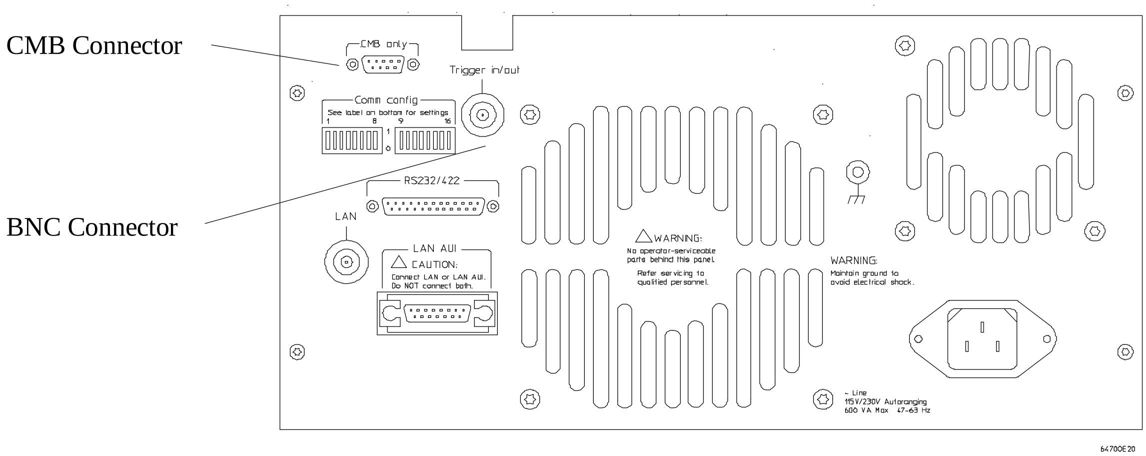
You can use the analyzer in one HP 64700 to arm (that is, activate) the analyzers in other HP 64700 Card Cages or to cause emulator execution in other HP 64700 Card Cages to break into the monitor.

You can use the HP 64700’s BNC connector (labeled TRIGGER IN/OUT on the lower left corner of the HP 64700 rear panel) to trigger an external instrument (for example, a logic analyzer or oscilloscope) when the analyzer finds its trigger condition. Also, you can allow an external instrument to arm the analyzer or break emulator execution into the monitor.

The coordinated measurement tasks you can perform are grouped into the following sections:

- Setting up for coordinated measurements.
- Starting and stopping multiple emulators.
- Driving trigger signals to the CMB or BNC.
- Stopping program execution on trigger signals.
- Arming analyzers on trigger signals.

The location of the CMB and BNC connectors on the HP 64700 rear panel is shown in the following figure.



Signal Lines on the CMB

There are three bi-directional signal lines on the CMB connector on the rear panel of the emulator. These CMB signals are:

TRIGGER The CMB TRIGGER line is low true. This signal can be driven or received by any HP 64700 connected to the CMB. This signal can be used to trigger an analyzer. It can be used as a break source for the emulator.

READY The CMB READY line is high true. It is an open collector and performs an ANDing of the ready state of enabled emulators on the CMB. Each emulator on the CMB releases this line when it is ready to run. This line goes true when all enabled emulators are ready to run, providing for a synchronized start.

When CMB is enabled, each emulator is required to break to background when CMB READY goes false, and will wait for CMB READY to go true before returning to the run state. When an enabled emulator breaks, it will drive the CMB READY false and will hold it false until it is ready to resume running. When an emulator is reset, it also drives CMB READY false.

EXECUTE The CMB EXECUTE line is low true. Any HP 64700 on the CMB can drive this line. It serves as a global interrupt and is processed by both the emulator and the analyzer. This signal causes an emulator to run from a specified address when CMB READY returns true.

BNC Trigger Signal

The BNC trigger signal is a positive rising edge TTL level signal. The BNC trigger line can be used to either drive or receive an analyzer trigger, or receive a break request for the emulator.

Comparison Between CMB and BNC Triggers The CMB trigger and BNC trigger lines have the same logical purpose: to provide a means for connecting the internal trigger signals (trig1 and trig2) to external instruments. The CMB and BNC trigger lines are bi-directional. Either signal may be used directly as a break condition.

The CMB trigger is level-sensitive, while the BNC trigger is edge-sensitive. The CMB trigger line puts out a true pulse following receipt of EXECUTE, despite the commands used to configure it. This pulse is internally ignored.

Note that if you use the EXECUTE function, the CMB TRIGGER should not be used to trigger external instruments, because a false trigger will be generated when EXECUTE is activated.

Setting Up for Coordinated Measurements

This section describes how to:

- Connect the Coordinated Measurement Bus.
- Connect the rear panel BNC.

To connect the Coordinated Measurement Bus (CMB)

Caution

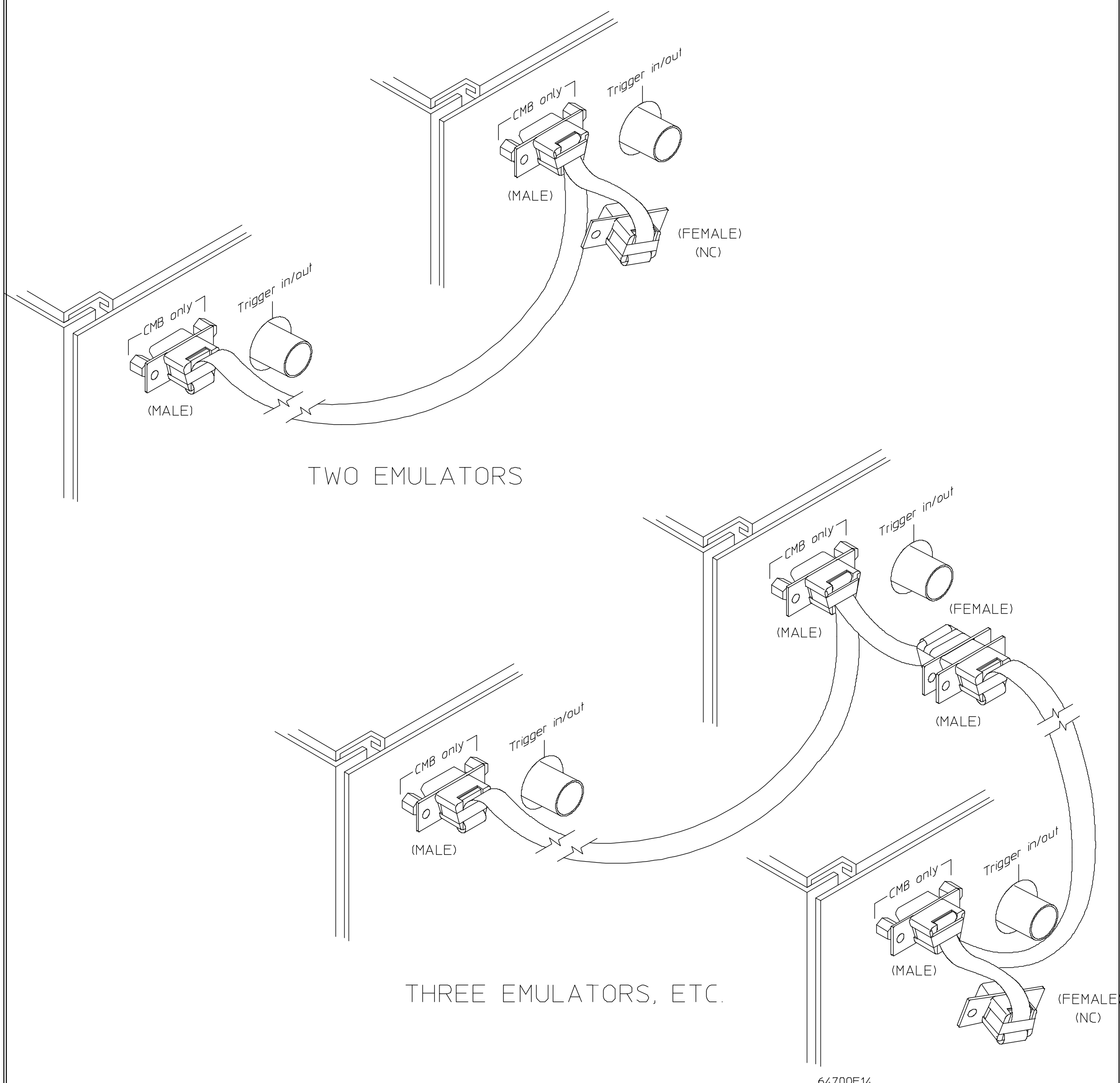
Be careful not to confuse the 9-pin connector used for CMB with those used by some computer systems for RS-232C communications. Applying RS-232C signals to the CMB connector is likely to result in damage to the HP 64700 Card Cage.

To use the CMB, you will need one CMB cable for the first two emulators and one additional cable for every emulator after the first two. The CMB cable is orderable from HP under product number HP 64023A. The cable is four meters long.

You can build your own compatible CMB cables using standard 9-pin D type subminiature connectors and 26 AWG wire.

Note that Hewlett-Packard does not ensure proper CMB operation if you are using a self-built cable!

1 Connect the cables to the HP 64700 CMB ports.



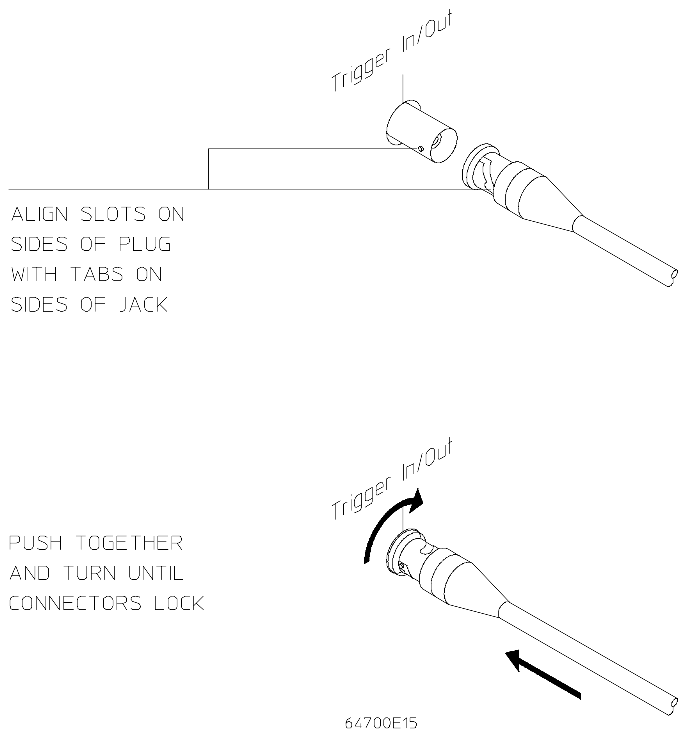
Number of HP 64700 Series Emulators	**Maximum Total Length of Cable**	**Restrictions on the CMB Connection**
2 to 8	100 meters	None.
9 to 16	50 meters	None.
9 to 16	100 meters	Only 8 emulators may have rear panel pullups connected. *
17 to 32	50 meters	Only 16 emulators may have rear panel pullups connected. *
* A modification must be performed by your HP Customer Engineer. Emulators using the CMB must use background emulation monitors. At least 3/4 of the HP 64700-Series emulators connected to the CMB must be powered up before proper operation of the entire CMB configuration can be assured.		

To connect to the rear panel BNC

Caution

The BNC line on the HP 64700 accepts input and output of TTL levels only. (TTL levels should not be less than 0 volts or greater than 5 volts.) Failure to observe these specifications may result in damage to the HP 64700 Card Cage.

1 Connect one end of a 50 ohm coaxial cable with male BNC connectors to the HP 64700 BNC receptacle and the other end to the appropriate BNC receptacle on the other measuring instrument.



The BNC connector is capable of driving TTL level signals into a 50 ohm load. (A positive rising edge is the trigger signal.) It requires a driver that can supply at least 4 mA at 2 volts when used as a receiver. The BNC connector is configured as an open-emitter structure which allows for multiple drivers to be connected. It can be used for cross-triggering between multiple HP 64700Bs when no other cross-measurements are needed. The output of the BNC connector is short-circuit protected and is protected from TTL level signals when the emulator is powered down.

Starting/Stopping Multiple Emulators

When HP 64700 Card Cages are connected together via the Coordinated Measurement Bus (CMB), you can start and stop up to 32 emulators at the same time. These are called synchronous measurements.

This section describes how to:

- Enable synchronous measurements.
- Start synchronous measurements.
- Disable synchronous measurements.

To enable synchronous measurements

- Enter the **specify run** command.

You can enable the emulator's interaction with the CMB by using the **specify run** command. When the EXECUTE signal is received, the emulator will run at the current program counter address or the address specified in the **specify run** command.

Note that when the CMB is being actively controlled by another emulator, the **step** command does not work correctly. The emulator may end up running in user code (NOT stepping). Disable CMB interaction (see "To disable synchronous measurements" below) while stepping the processor.

Note that enabling CMB interaction does not affect the operation of analyzer cross-triggering.

You can use the **specify trace** command to specify that an analyzer measurement begin upon reception of the CMB EXECUTE signal.

The trace measurement defined by the **specify trace** command will be started when the EXECUTE signal becomes active. When the trace measurement begins, you will see the message "CMB execute; emulation trace started".

When you enter a normal **trace** command, trace at execute is disabled, and the analyzer ignores the CMB EXECUTE signal.

Examples

To enable synchronous measurements:

specify run from `1e8h <RETURN>`

To trace when synchronous execution begins:

specify trace after address `main <RETURN>`

To start synchronous measurements

- Enter the **cmb_execute** command.

The **cmb_execute** command will cause the EXECUTE line to be pulsed, thereby initiating a synchronous measurement. CMB interaction does not have to be enabled in order to use either of these commands. (When you enable CMB interaction, you only specify how the emulator will react to the CMB EXECUTE signal.)

All emulators whose CMB interaction is enabled will break into the monitor when any one of those emulators breaks into its monitor.

To disable synchronous measurements

- Enter the **specify run disable** command.

You can disable the emulator's interaction with the CMB by using the **specify run disable** command. When interaction is disabled, the emulator ignores the CMB EXECUTE and READY lines.

Using Trigger Signals

The HP 64700 contains two internal lines, trig1 and trig2, over which trigger signals can pass from the emulator or analyzer to other HP 64700s on the Coordinated Measurement Bus (CMB) or other instruments connected to the BNC connector.

You can configure the internal lines to make connections between the emulator, analyzer, CMB connector, or BNC connector. Measurements that depend on these connections are called *interactive measurements* or *coordinated measurements*.

To configure the internal trig1 and trig2 lines, you must enter the **modify configuration** command and then answer "yes" to the "Modify interactive measurement specification?" question. When you do this, the following display appears.

```
                     Interactive Measurement Specification

            BNC <<-??->> ---\                      BNC <<-??->> ---\
                            |                                      |
           CMBT <<-??->> ---|                     CMBT <<-??->> ---|
                            | Trig1                                | Trig2
       Emulator <<------ ---|                 Emulator <<-??--- ---|
                            |                                      |
       Analyzer ------>> ---/                 Analyzer <<-??->> ---/

NOTES:
  1. The connections marked "??" are set up here in configuration.
  2. drive = ---->>  receive = <<----  (The display won't change, however.)
```

This display illustrates the possible connections between the internal lines (trig1 and trig2) and the emulator, analyzer, and external devices.

Notice that the analyzer always drives trig1, and the emulator always receives trig1. This provides for the **break_on_trigger** syntax of the **trace** command.

You can use the trig1 or trig2 line to make a connection between the analyzer and the CMB connector or BNC connector so that, when the analyzer finds its trigger condition, a trigger signal is driven on the HP 64700's Coordinated Measurement Bus (CMB) or BNC connector.

You can use the trig1 or trig2 line to make a connection between the emulator break input and the CMB connector, BNC connector, or analyzer so that program

execution can break when a trigger signal is received from the CMB, BNC, or analyzer.

You can use the trig2 line to make a connection between the analyzer and the CMB connector or BNC connector so that the analyzer can be armed (that is, enabled) when a trigger signal is received from the CMB or BNC connector.

You can use the trig1 and trig2 lines to make several type of connections at the same time. For example, when the analyzer finds its trigger condition, a signal is driven on the trig1 line. This signal may be used to stop user program execution, but the trigger signal may also be driven on the CMB and BNC connectors.

Also, it's possible for signals to be driven and received on the CMB or BNC connectors. So, for example, while the analyzer's trigger signal can be driven on the CMB and BNC connectors, signals can also be received from the CMB and BNC connectors and used to stop user program execution. In this case, the emulator will break into the monitor on either the analyzer trigger or on the reception of a trigger signal from the CMB or BNC.

You can disable connections made by the internal trig1 and trig2 lines by answering "neither" or "no" to the appropriate interactive measurement configuration question.

This section shows you how to:

- Drive the emulation analyzer trigger signal to the CMB.
- Drive the emulation analyzer trigger signal to the BNC connector.
- Break emulator execution on signal from CMB.
- Break emulator execution on signal from BNC.
- Arm the emulation analyzer on signal from CMB.
- Arm the emulation analyzer on signal from BNC.

To drive the emulation analyzer trigger signal to the CMB

1 Enter the **modify configuration** command.

2 Answer "yes" to the "Modify interactive measurement specification?" question.

3 Answer "receive" to the "Should CMBT drive or receive Trig1?" question.

You could also drive the emulation analyzer trigger to the CMB over the trig2 internal line by specifying that the CMBT should receive trig2 and that the emulation analyzer should drive trig2.

To drive the emulation analyzer trigger signal to the BNC connector

1. Enter the **modify configuration** command.
2. Answer "yes" to the "Modify interactive measurement specification?" question.
3. Answer "receive" to the "Should BNC drive or receive Trig1?" question.

You could also drive the emulation analyzer trigger to the BNC over the trig2 internal line by specifying that the BNC should receive trig2 and that the emulation analyzer should drive trig2.

To break emulator execution on signal from CMB

1. Enter the **modify configuration** command.
2. Answer "yes" to the "Modify interactive measurement specification?" question.
3. Answer "drive" to the "Should CMBT drive or receive Trig1?" question.

You could also break emulator execution on a trigger signal from the CMB over the trig2 internal line by specifying that the CMB should drive trig2 and that the emulator break should receive trig2.

To break emulator execution on signal from BNC

1. Enter the **modify configuration** command.
2. Answer "yes" to the "Modify interactive measurement specification?" question.
3. Answer "drive" to the "Should BNC drive or receive Trig1?" question.

   You could also break emulator execution on a trigger signal from the BNC over the trig2 internal line by specifying that the BNC should drive trig2 and that the emulator break should receive trig2.

To arm the emulation analyzer on signal from CMB

1. Enter the **modify configuration** command.
2. Answer "yes" to the "Modify interactive measurement specification?" question.
3. Answer "drive" to the "Should CMBT drive or receive Trig2?" question.
4. Answer "receive" to the "Should Analyzer drive or receive Trig2?" question.
5. Use the **arm_trig2** option to the **trace** command.

To arm the emulation analyzer on signal from BNC

1 Enter the **modify configuration** command.

2 Answer "yes" to the "Modify interactive measurement specification?" question.

3 Answer "drive" to the "Should BNC drive or receive Trig2?" question.

4 Answer "receive" to the "Should Analyzer drive or receive Trig2?" question.

5 Use the **arm_trig2** option to the **trace** command.

10

Setting X Resources

Setting X Resources

The Graphical User Interface is an X Window System application which means it is a *client* in the X Window System client-server model.

The X server is a program that controls all access to input devices (typically a mouse and a keyboard) and all output devices (typically a display screen). It is an interface between application programs you run on your system and the system input and output devices.

An X *resource* controls an element of appearance or behavior in an X application. For example, in the graphical interface, one resource controls the text in action key pushbuttons as well as the action performed when the pushbutton is clicked.

By modifying resource settings, you can change the appearance or behavior of certain elements in the graphical interface.

When the graphical interface starts up, it reads resource specifications from a set of configuration files. Resources specifications in later files override those in earlier files. Files are read in the following order:

1 The application defaults file. For example, /usr/lib/X11/app-defaults/HP64_Softkey when the operating system is HP-UX or /usr/openwin/lib/X11/app-defaults/HP64_Softkey when the operating system is SunOS.

2 The $XAPPLRESDIR/HP64_Softkey file. (The XAPPLRESDIR environment variable defines a directory containing system-wide custom application defaults.)

3 The server's RESOURCE_MANAGER property. (The **xrdb** command loads user-defined resource specifications into the RESOURCE_MANAGER property.)

   If no RESOURCE_MANAGER property exists, user defined resource settings are read from the $HOME/.Xdefaults file.

4 The file named by the XENVIRONMENT environment variable.

   If the XENVIRONMENT variable is not set, the $HOME/.Xdefaults-*host* file (typically containing resource specifications for a specific remote host) is read.

5 Resource specifications included in the command line with the **-xrm** option.

6 System scheme files in directory /usr/hp64000/lib/X11/HP64_schemes.

7 System-wide custom scheme files located in directory $XAPPLRESDIR/HP64_schemes.

8 User-defined scheme files located in directory $HOME/.HP64_schemes (note the dot in the directory name).

Scheme files group resource specifications for different displays, computing environments, and languages.

This chapter shows you how to:

- Modify the Graphical User Interface resources.
- Use customized scheme files.
- Set up custom action keys.
- Set initial recall buffer values.
- Set up demos or tutorials.

Refer to the "X Resources and the Graphical Interface" section in the "Concepts" chapter for more detailed information.

To modify the Graphical User Interface resources

You can customize the appearance of an X Windows application by modifying its X resources. The following tables describe some of the commonly modified application resources.

Application Resources for Schemes

Resource	**Values**	**Description**
HP64_Softkey.platformScheme	HP-UX SunOS (custom)	Names the subdirectory for platform specific schemes. This resource should be set to the platform on which the X server is running (and displaying the Graphical User Interface) if it is different than the platform where the application is running.
HP64_Softkey.colorScheme	BW Color (custom)	Names the color scheme file.
HP64_Softkey.sizeScheme	Small Large (custom)	Names the size scheme file which defines the fonts and the spacing used.
HP64_Softkey.labelScheme	Label $LANG (custom)	Names to use for labels and button text. The default uses the $LANG environment variable if it is set and if a scheme file named Softkey.$LANG exists in one of the directories searched for scheme files; otherwise, the default is Label.
HP64_Softkey.inputScheme	Input (custom)	Specifies mouse and keyboard operation.

Commonly Modified Application Resources

Resource	Values	Description
HP64_Softkey.lines	24 (min. 18)	Specifies the number of lines in the main display area.
HP64_Softkey.columns	100 (min. 80)	Specifies the number of columns, in characters, in the main display area.
HP64_Softkey.enableCmdline	True False	Specifies whether the command line area is displayed when you initially enter the Graphical User Interface.
*editFile	(example) vi %s	Specifies the command used to edit files.
*editFileLine	(example) vi +%d %s	Specifies the command used to edit a file at a certain line number.
*<proc>*actionKeysSub.keyDefs	(paired list of strings)	Specifies the text that should appear on the action key push buttons and the commands that should be executed in the command line area when the action key is pushed. Refer to the "To set up custom action keys" section for more information.
*<proc>*dirSelectSub.entries	(list of strings)	Specifies the initial values that are placed in the **File→Context→Directory** popup recall buffer. Refer to the "To set initial recall buffer values" section for more information.
*<proc>*recallSub.entries	(list of strings)	Specifies the initial values that are placed in the entry buffer (labeled "():"). Refer to the "To set initial recall buffer values" section for more information.

The following steps show you how to modify the Graphical User Interface’s X resources.

1 Copy part or all of the HP64_Softkey application defaults file to a temporary file.

The HP64_Softkey file contains the default definitions for the graphical interface application’s X resources.

For example, on an HP 9000 computer you can use the following command to copy the complete HP64_Softkey file to HP64_Softkey.tmp (note that the HP64_Softkey file is several hundred lines long):

cp /usr/lib/X11/app-defaults/HP64_Softkey HP64_Softkey.tmp

NOTE: The HP64_Softkey application defaults file is re-created each time Graphical User Interface software is installed or updated. You can use the UNIX **diff** command to check for differences between the new HP64_Softkey application defaults file and the old application defaults file that is saved as /usr/hp64000/lib/X11/HP64_schemes/old/HP64_Softkey.

2 Modify the temporary file.

Modify the resource that defines the behavior or appearance that you wish to change.

For example, to change the number of lines in the main display area to 36:

vi HP64_Softkey.tmp

Search for the string "HP64_Softkey.lines". You should see lines similar to the following.

```
!-----------------------------------------------------------------------------
! The lines and columns set the vertical and horizontal dimensions of the
! main display area in characters, respectively.  Minimum values are 18 lines
! and 80 columns.  These minimums are silently enforced.
!
! Note: The application cannot be resized by using the window manager.

!HP64_Softkey.lines:    24
!HP64_Softkey.columns:  85
```

Edit the line containing "HP64_Softkey.lines" so that it is uncommented and is set to the new value:

```
!--------------------------------------------------------------------------
! The lines and columns set the vertical and horizontal dimensions of the
! main display area in characters, respectively.  Minimum values are 18 lines
! and 80 columns.  These minimums are silently enforced.
!
! Note: The application cannot be resized by using the window manager.

HP64_Softkey.lines:    36
!HP64_Softkey.columns:  85
```

Save your changes and exit the editor.

3 If the RESOURCE_MANAGER property exists (as is the case with HP VUE — if you're not sure, you can check by entering the **xrdb -query** command), use the **xrdb** command to add the resources to the RESOURCE_MANAGER property. For example:

xrdb -merge -nocpp HP64_Softkey.tmp

Otherwise, if the RESOURCE_MANAGER property does not exist, append the temporary file to your $HOME/.Xdefaults file. For example:

cat HP64_Softkey.tmp >> $HOME/.Xdefaults

4 Remove the temporary file.

5 Start or restart the Graphical User Interface.

After you have completed the above steps, you must either start, or restart by exiting and starting again, the Graphical User Interface. Starting and exiting the Graphical User Interface is described in the "Using the Interface" chapter.

To use customized scheme files

Scheme files are used to set platform specific resources that deal with color, fonts and sizes, mouse and keyboard operation, and labels and titles. You can create and use customized scheme files by following these steps.

1 Create the $HOME/.HP64_schemes/<platform> directory.

For example:

mkdir $HOME/.HP64_schemes
mkdir $HOME/.HP64_schemes/HP-UX

2 Copy the scheme file to be modified to the $HOME/.HP64_schemes/<platform> directory.

Label scheme files are not platform specific; therefore, they should be placed in the $HOME/.HP64_schemes directory. All other scheme files should be placed in the $HOME/.HP64_schemes/<platform> directory.

For example:

cp /usr/hp64000/lib/X11/HP64_schemes/HP-UX/Softkey.Color $HOME/.HP64_schemes/HP-UX/Softkey.MyColor

Note that if your custom scheme file has the same name as the default scheme file, the load order requires resources in the custom file to explicitly override resources in the default file.

3 Modify the $HOME/.HP64_schemes/<platform>/Softkey.<scheme> file.

For example, you could modify the "$HOME/.HP64_schemes/HP-UX/Softkey.MyColor" file to change the defined foreground and background colors. Also, since the scheme file name is different than the default, you could comment out various resource settings to cause general foreground and background color definitions to apply to the Graphical User Interface. At least one resource must be defined in your color scheme file for it to be recognized.

4 If your custom scheme file has a different name than the default, you must modify the scheme resource definitions.

The Graphical User Interface application defaults file contains resources that specify which scheme files are used. If your custom scheme files are named differently than the default scheme files, you must modify these resource settings so that your customized scheme files are used instead of the default scheme files.

For example, to use the "$HOME/.HP64_schemes/HP-UX/Softkey.MyColor" color scheme file you would set the "HP64_Softkey.colorScheme" resource to "MyColor":

```
HP64_Softkey.colorScheme: MyColor
```

Refer to the previous "To customize Graphical User Interface resources" section for more detailed information on modifying resources.

To set up custom action keys

- Modify the "actionKeysSub.keyDefs" resource.

The "actionKeysSub.keyDefs" resource defines a list of paired strings. The first string defines the text that should appear on the action key pushbutton. The second string defines the command that should be sent to the command line area and executed when the action key is pushed.

A pair of parentheses (with no spaces, that is "()") can be used in the command definition to indicate that text from the entry buffer should replace the parentheses when the command is executed.

Action keys that use the entry buffer should always include the entry buffer symbol, "( )", in the action key label as a visual cue to remind you to place information in the entry buffer before clicking the action key.

Shell commands can be executed by using an exclamation point prefix. A second exclamation point ends the command string and allows additional options on the command line.

Also, command files can be executed by placing the name of the file in the command definition.

Finally, an empty action ("") means to repeat the previous operation, whether it came from a pulldown, a dialog, a popup, or another action key.

Examples

To set up custom action keys when the graphical interface is used with the 68340 emulator, modify the "*m68340*actionKeysSub.keyDefs" resource:

```
*m68340*actionKeysSub.keyDefs: \
   "Make"              "cd /users/project2/68340; !make! in_browser" \
   "Load Pgm"          "load configuration config.EA; load program2" \
   "Run Pgm"           "run from reset" \
   "Trace after ( )"   "trace after (); display trace" \
   "Step Source"       "set source on; display memory mnemonic; step source" \
   "Again"             ""
```

Refer to the previous "To modify Graphical User Interface resources" section for more detailed information on modifying resources.

To set initial recall buffer values

- Modify the "entries" resource for the particular recall buffer.

There are six popup recall buffers present in the Graphical User Interface. The resources for these popup recall buffers are listed in the following table.

The window manager resource "*transientDecoration" controls the borders around dialog box windows. The most natural setting for this resource is "title."

<table>
<tr><th colspan="3">Popup Recall Buffer Resources</th></tr>
<tr><th>Recall Popup</th><th>Resources</th><th>Description</th></tr>
<tr><td>File→Context→Directory ...</td><td>*dirSelect.textColumns
*dirSelect.listVisibleItemCount
*dirSelectSub.entries</td><td rowspan="6">The default number of text columns in the popup is 50.

The default number of visible lines in the popup is 12.

The "entries" resource is defined as a list of strings (see the following example).

Up to 40 unique values are saved in each of the recall buffers (as specified by the resource settings "*XcRecall.maxDepth: 40" and "*XcRecall.onlyUnique: True").</td></tr>
<tr><td>File→Context→Symbols ...</td><td>*symSelect.textColumns
*symSelect.listVisibleItemCount
*symSelectSub.entries</td></tr>
<tr><td>Trace→Trace Spec ...</td><td>*modtrace.textColumns
*modtrace.listVisibleItemCount
*modtraceSub.entries</td></tr>
<tr><td>Entry Buffer ():</td><td>*recall.textColumns
*recall.listVisibleItemCount
*recallSub.entries</td></tr>
<tr><td>Command Line command recall</td><td>*recallCmd.textColumns
*recallCmd.listVisibleItemCount
*recallCmdSub.entries</td></tr>
<tr><td>Command Line pod/simio recall</td><td>*recallKbd.textColumns
*recallKbd.listVisibleItemCount
*recallKbdSub.entries</td></tr>
</table>

Examples

To set the initial values for the directory selection dialog box when the Graphical User Interface is used with 68340 emulators, modify the "*m68340*dirSelectSub.entries" resource:

```
*m68340*dirSelectSub.entries: \
   "$HOME" \
   ".." \
   "/users/project1" \
   "/users/project2/68340"
```

Refer to the previous "To modify the Graphical User Interface resources" section for more detailed information on modifying resources.

To set up demos or tutorials

You can add demos or tutorials to the Graphical User Interface by modifying the resources described in the following tables.

Demo Related Component Resources		
Resource	**Value**	**Description**
*enableDemo	False True	Specifies whether **Help→Demo** appears in the pulldown menu.
*demoPopupSub.indexFile	./Xdemo/Index-topics	Specifies the file containing the list of topic and file pairs.
*demoPopup.textColumns	30	Specifies the width, in characters, of the of the demo topic list popup.
*demoPopup.listVisibleItemCount	10	Specifies the length, in lines, of the demo topic list popup.
*demoTopic	About demos	Specifies the default topic in the demo popup selection buffer.

Tutorial Related Component Resources		
Resource	**Value**	**Description**
*enableTutorial	False True	Specifies whether **Help→Tutorial** appears in the pulldown menu.
*tutorialPopupSub.indexFile	./Xtutorial/Index-topics	Specifies the file containing the list of topic and file pairs.
*tutorialPopup.textColumns	30	Specifies the width, in characters, of the of the tutorial topic list popup.
*tutorialPopup.listVisibleItemCount	10	Specifies the length, in lines, of the tutorial topic list popup.
*tutorialTopic	About tutorials	Specifies the default topic in the tutorial popup selection buffer.

The mechanism for providing demos and tutorials in the graphical interface is identical. The following steps show you how to set up demos or tutorials in the Graphical User Interface.

1 Create the demo or tutorial topic files and the associated command files.

Topic files are simply ASCII text files. You can use "\I" to produce inverse video in the text, "\U" to produce underlining in the text, and "\N" to restore normal text.

Command files are executed when the "Press to perform demo (or tutorial)" button (in the topic popup dialog) is pushed. A command file must have the same name as the topic file with ".cmd" appended. Also, a command file must be in the same directory as the associated topic file.

2 Create the demo or tutorial index file.

Each line in the index file contains first a quoted string that is the name of the topic which appears in the index popup and second the name of the file that is raised when the topic is selected. For example:

```
"About demos"       /users/guest/gui_demos/general
"Loading programs"  /users/guest/gui_demos/loadprog
"Running programs"  /users/guest/gui_demos/runprog
```

You can use absolute paths (for example, /users/guest/topic1), paths relative to the directory in which the interface was started (for example, mydir/topic2), or paths relative to the product directory (for example, ./Xdemo/general where the product directory is something like /usr/hp64000/inst/emul/64751A).

3 Set the "*enableDemo" or "*enableTutorial" resource to "True".

4 Define the demo index file by setting the "*demoPopupSub.indexFile" or "*tutorialPopupSub.indexFile" resource.

For example:

```
*demoPopupSub.indexFile:  /users/guest/gui_demos/index
```

You can use absolute paths (for example, /users/guest/Index), paths relative to the directory in which the interface was started (for example, mydir/indexfile), or paths relative to the product directory (for example, ./Xdemo/Index-topics where the product directory is something like /usr/hp64000/inst/emul/64751A).

5 If you wish to define a default topic to be selected, set the "*demoTopic" or "*tutorialTopic" resource to the topic string.

For example:

```
*demoTopic:  "About demos"
```

Refer to the previous "To customize Graphical User Interface resources" section for more detailed information on modifying resources.

Part 3

Reference

Descriptions of the product in a dictionary or encyclopedia format.

Part 3

11

Emulator/Analyzer Interface Commands

Emulator/Analyzer Interface Commands

This chapter describes the emulator/analyzer interface commands in alphabetical order. First, the syntax conventions are described and the commands are summarized.

How Pulldown Menus Map to the Command Line

The following table shows the items available in the pulldown menus and the command line commands to which they map.

Pulldown	Command Line
File→Context→Directory	cd
File→Context→Symbols	cws
File→Load→Emulator Config	load configuration
File→Load→Executable	load <abs_file>
File→Load→Program Only	load <abs_file> nosymbols
File→Load→Symbols Only	load symbols
File→Store→Trace Data	store trace
File→Store→Trace Spec	store trace_spec
File→Store→BBA Data	bbaunload
File→Copy→Display	copy display to
File→Copy→Memory	copy memory to
File→Copy→Data Values	copy data to
File→Copy→Trace	copy trace to
File→Copy→Registers	copy registers to
File→Copy→Breakpoints	copy software_breakpoints to
File→Copy→Status	copy status to
File→Copy→Global Symbols	copy global_symbols to
File→Copy→Local Symbols ()	copy local_symbols_in **--**SYMB**--** to
File→Copy→Pod Commands	copy pod_command to
File→Copy→Error Log	copy error_log to
File→Copy→Event Log	copy event_log to
File→Log→Playback	<command file>
File→Log→Record	log_commands to
File→Log→Stop	log_commands off
File→Emul700→High-Level Debugger	N/A
File→Emul700→Performance Analyzer	N/A
File→Emul700→Emulator/Analyzer	N/A
File→Emul700→Timing Analyzer	N/A
File→Edit→File	! vi <file> ! no_prompt_before_exit
File→Edit→At () Location	! vi +<line> <file> ! no_prompt_before_exit
File→Edit→At PC Location	! vi +<line> <file> ! no_prompt_before_exit
File→Term	!
File→Exit→Window (save session)	end
File→Exit→Locked (all windows, save session)	end locked
File→Exit→Released (all windows, release emulator)	end release_system

Pulldown	Command Line
Display→Context	pwd, pws
Display→Memory	display memory
Display→Memory→Mnemonic ()	display memory **--**EXPR**--** mnemonic
Display→Memory→Mnemonic at PC	display memory mnemonic at_pc
Display→Memory→Mnemonic Previous	display memory mnemonic previous_display
Display→Memory→Hex ()→bytes	display memory **--**EXPR**--** blocked bytes
Display→Memory→Hex ()→words	display memory **--**EXPR**--** blocked words
Display→Memory→Hex ()→long	display memory **--**EXPR**--** blocked long
Display→Memory→Real ()→short	display memory **--**EXPR**--** real short
Display→Memory→Real ()→long	display memory **--**EXPR**--** real long
Display→Memory→At ()	display memory **--**EXPR**--**
Display→Memory→Repetitively	display memory repetitively
Display→Data Values	display data
Display→Data Values→New ()→<type>	display data **--**EXPR**--** <type>
Display→Data Values→Add ()→<type>	display data, **--**EXPR**--** <type>
Display→Trace	display trace
Display→Registers	display registers
Display→Breakpoints	display software_breakpoints
Display→Status	display status
Display→Simulated IO	display simulated_io
Display→Global Symbols	display global_symbols
Display→Local Symbols ()	display local_symbols_in **--**SYMB**--**
Display→Pod Commands	display pod_command
Display→Error Log	display error_log
Display→Event Log	display event_log
Modify→Emulator Config	modify configuration
Modify→Memory	modify memory
Modify→Memory at ()	modify memory **--**EXPR**--**
Modify→Register	modify register

Pulldown	Command Line
Execution→Run→from PC	run
Execution→Run→from ()	run from --EXPR--
Execution→Run→from Transfer Address	run from transfer_address
Execution→Run→from Reset	run from reset
Execution→Run→until ()	run until --EXPR--
Execution→Step Source→from PC	step source
Execution→Step Source→from ()	step source from --EXPR--
Execution→Step Source→from Transfer Address	step source from transfer_address
Execution→Step Instruction→from PC	step
Execution→Step Instruction→from ()	step from --EXPR--
Execution→Step Instruction→from Transfer Address	step from transfer_address
Execution→Break	break
Execution→Reset	reset
Breakpoints→Display	display software_breakpoints
Breakpoints→Enable	modify software_breakpoints enable/disable
Breakpoints→Permanent ()	modify software_breakpoints set --EXPR-- permanent
Breakpoints→Temporary ()	modify software_breakpoints set --EXPR-- temporary
Breakpoints→Set All	modify software_breakpoints set
Breakpoints→Clear ()	modify software_breakpoints clear --EXPR--
Breakpoints→Clear All	modify software_breakpoints clear
Trace→Display	display trace
Trace→Trace Spec	N/A (browses recall buffer for trace commands)
Trace→After ()	trace after STATE
Trace→Before ()	trace before STATE
Trace→About ()	trace about STATE
Trace→Only ()	trace only STATE
Trace→Only () Prestore	trace only STATE prestore anything
Trace→Again	trace again
Trace→Repetitively	<previous trace spec> repetitively
Trace→Everything	trace
Trace→Until ()	trace before STATE break_on_trigger
Trace→Until Stop	trace on_halt
Trace→Stop	stop_trace

Pulldown	Command Line
Settings→Source/Symbol Modes→Absolute	set source off symbols off
Settings→Source/Symbol Modes→Symbols	set source off symbols on
Settings→Source/Symbol Modes→Source Mixed	set source on inverse_video on symbols on
Settings→Source/Symbol Modes→Source Only	set source only inverse_video off symbols on
Settings→Display Modes→Source Only	set
Settings→Pod Command Keyboard	display pod_command; pod_command keyboard
Settings→Simulated IO Keyboard	display simulated_io; modify keyboard_to_simio
Settings→Command Line	N/A (toggles the command line)

How Popup Menus Map to the Command Line

The following tables show the items available in the popup menus and the command line commands to which they map.

Mnemonic Memory Display Popup	Command Line
Set/Clear Breakpoint	modify software_breakpoints set/clear --EXPR--
Edit Source	! vi +<line> <file> ! no_prompt_before_exit
Run Until	run until --EXPR--
Trace After	trace after STATE
Trace Before	trace before STATE
Trace About	trace about STATE
Trace Until	trace before STATE break_on_trigger

Breakpoints Display Popup	Command Line
Set/Inactivate Breakpoint	modify software_breakpoints set/deactivate --EXPR--
Clear (delete) Breakpoint	modify software_breakpoints clear --EXPR--
Enable/Disable Software Breakpoints	modify software_breakpoints enable/disable
Set All Breakpoints	modify software_breakpoints set
Clear (delete) All Breakpoints	modify software_breakpoints clear

Symbols Display Popup	Command Line
Display Local Symbols	display local_symbols_in --SYMB--
Display Parent Symbols	display local_symbols_in --SYMB--, display global_symbols
Cut Full Symbol Name	N/A
Edit File Defining Symbol	! vi +<line> <file> ! no_prompt_before_exit

Status Line Popup	Command Line
Remove Temporary Message	N/A
Display Error Log	display error_log
Display Event Log	display event_log
Command Line On/Off	(toggles command line)

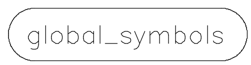
Command Line Popup	Command Line
Position Cursor, Replace Mode	<INSERT CHAR> key (when in insert mode)
Position Cursor, Insert Mode	<INSERT CHAR> key
Execute Command	<RETURN> key
Clear to End of Line	<CTRL>e
Clear Entire Line	<CTRL>u
Command Line Off	(toggles command line)

Syntax Conventions

Conventions used in the command syntax diagrams are defined below.

Oval-shaped Symbols

Oval-shaped symbols show options available on the softkeys and other commands that are available, but do not appear on softkeys (such as **log_commands** and **wait**). These appear in the syntax diagrams as:

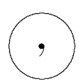


Rectangular-shaped Symbols

Rectangular-shaped symbols contain prompts or references to other syntax diagrams. Prompts are enclosed with angle brackets (< and >). References to other diagrams are shown in all capital letters. Also, references to expressions are shown in all capital letters, for example **--**EXPR**--** and **--**SYMB**--** (see those syntax diagrams). These appear in the following syntax diagrams as:

Circles

Circles indicate operators and delimiters used in expressions and on the command line as you enter commands. These appear in the syntax diagrams as:



The -NORMAL- Key

The softkey labeled **-NORMAL-** allows you exit the **--**SYMB**--** definition, and access softkeys that are not displayed when defining expressions. You can press this key after you have defined an expression to view other available options.

Commands

Emulator/analyzer interface commands are summarized in the table below and described in the following pages.

!UNIX_COMMAND	display event_log	modify memory[4]
bbaunload	display global_symbols	modify register[1]
break	display local_symbols_in	modify software_breakpoints[1]
cd (change directory)[3]	display memory[4]	name_of_module[3]
cmb_execute	display pod_command	performance_measurement_end
<command file>[3]	display registers[1]	performance_measurement_init
copy data[4]	display simulated_io[2]	performance_measurement_run
copy display	display software_breakpoints	pod_command
copy error_log	display status	pwd (print working directory)[3]
copy event_log	display trace	pws (print working symbol)[3]
copy global_symbols	end	reset
copy help	forward	run
copy local_symbols_in	help[3]	set
copy memory[4]	load <absolute_file>	specify
copy pod_command	load configuration	step
copy registers[1]	load emul_mem	stop_trace
copy software_breakpoints	load trace	store memory
copy status	load trace_spec	store trace
copy trace	load user_memory	store trace_spec
cws(change working symbol)[3]	log_commands[3]	sync_sim_registers[1]
display data[4]	modify configuration	trace
display error_log	modify keyboard_to_simio[2]	wait[3]

[1] This option is not available in real-time mode.
[2] This is only available when simulated I/O is defined.
[3] These commands are not displayed on softkeys.
[4] This option is not available in real-time mode if addresses are in target system or single-port emulation memory.

bbaunld

This command is available when the HP Branch Validator product is installed. This basis branch analyzer (BBA) product is used to analyze the testing of your programs, create more complete test suites, and quantify your level of testing.

The HP Branch Validator records branches executed in a program and generates reports that provide information about program execution during testing. It uses a special C preprocessor to add statements that write to a data array when program branches are taken. After running the program in the emulator (using test input), you can use the **bbaunload** command to store the BBA information to a file. Then, you can generate reports based on the stored information.

See Also

Refer to the *HP Branch Validator (BBA) User's Guide* for complete details on the **bbaunload** command syntax.

break

This command causes the emulator to leave user program execution and begin executing in the monitor.

The behavior of **break** depends on the state of the emulator:

running	Break diverts the processor from execution of your program to the emulation monitor.
reset	Break releases the processor from reset, and diverts execution to the monitor.
running in monitor	The **break** command does not perform any operation while the emulator is executing in the monitor.

See Also

The **reset**, **run**, and **step** commands.

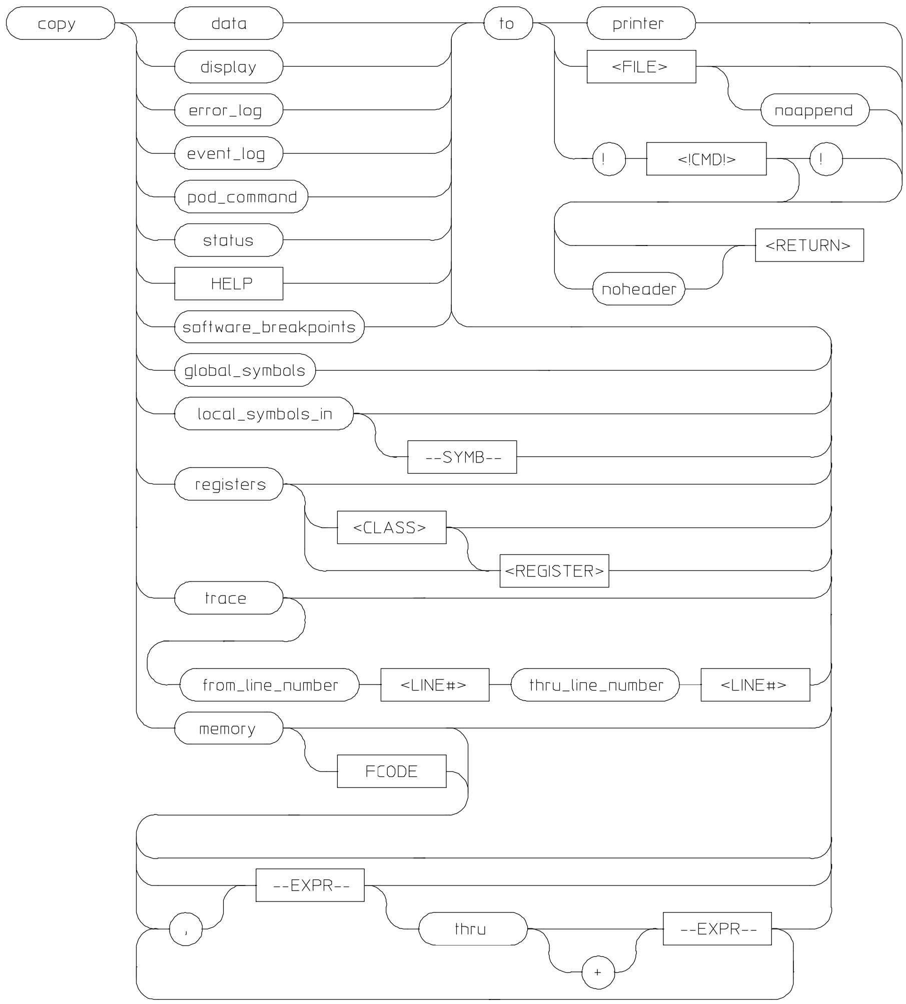
cmb_execute

The **cmb_execute** command causes the emulator to emit an EXECUTE pulse on its rear panel Coordinated Measurement Bus (CMB) connector. All emulators connected to the CMB (including the one sending the CMB EXECUTE pulse) and configured to respond to this signal will take part in the measurement.

See Also

The **specify run** and **specify trace** commands.

copy



Use this command with various parameters to save or print emulation and analysis information.

The **copy** command copies selected information to your system printer or listing file, or directs it to a UNIX process.

Depending on the information you choose to copy, default values may be options selected for the previous execution of the **display** command. For example, if you display memory locations 10h through 20h, then issue a **copy memory to myfile** command, myfile will list only memory locations 10h through 20h.

The parameters are as follows:

,	A comma used immediately after **memory** in the command line appends the current **copy memory** command to the preceding **display memory** command. The data specified in both commands is copied to the destination specified in the current command. Data is formatted as specified in the current command. The comma is also used as a delimiter between values when specifying multiple memory addresses.
<CLASS>	Specifies a particular class of the emulator registers. Refer to the "Accessing Registers" section in the "Using the Emulator" chapter for a list of the register names and classes.
<!CMD!>	This represents an UNIX filter or pipe where you want to route the output of the **copy** command. UNIX commands must be preceded by an exclamation point (!). An exclamation point following the UNIX command continues command line execution after the UNIX command executes. Emulation is not affected when using an UNIX command that is a shell intrinsic.
data	This allows you to copy a list of memory contents formatted in various data types (see display data).
display	This allows you to copy the display to a selected destination.
error_log	This allows you to copy the most recent errors that occurred.
event_log	This allows you to copy the most recent events that occurred.
!	An exclamation point specifies the delimiter for UNIX commands. An exclamation point must precede all UNIX commands. A trailing exclamation point should be used if you want to return to the command line and specify noheader. Otherwise, the trailing exclamation point is optional. If an exclamation point is part of the UNIX command, a backslash (\) must precede the exclamation point.
--EXPR--	An expression is a combination of numeric values, symbols, operators, and parentheses, specifying a memory address or offset value. See the EXPR syntax diagram.

FCODE

The function code used to define the address space being referenced. See the syntax diagram for FCODE to see a list of the function codes available and for an explanation of those codes.

<FILE>

This prompts you for the name of a file where you want the specified information to be copied. If you want to specify a file name that begins with a number, you must precede the file name with a backslash. For example: **copy display to \12.10 <RETURN>**

from_line_number

This specifies the trace list line number from which copying will begin.

global_symbols

This lets you copy a list of global symbols to the selected destination.

help

This allows you to copy the contents of the emulation help files to the selected destination.

<LINE#>

Use this with **from_line_number** and **thru_line_number** to specify the starting and ending trace list lines to be copied.

local_symbols_in

This lets you copy all the children of a given symbol to the selected destination. See the **--SYMB--** syntax page and the *Symbolic Retrieval Utilities User's Guide* for information on symbol hierarchy.

memory

This command copies the contents of a memory location or series of locations to the specified output. The memory contents are copied in the same format as specified in the last display memory command.

Contents of memory can be displayed if program runs are not restricted to real-time. Memory contents are listed as an asterisk (*) under the following conditions:

1 The address refers to guarded memory.

2 Runs are restricted to real-time, the emulator is running a user program, and the address is located in target system memory or in single-port emulation memory.

Initial values are the same as those specified by the command **display memory 0 blocked bytes offset_by 0**. Defaults are to values specified in the previous **display memory** command.

noappend

This causes any copied information to overwrite an existing file with the same name specified by <FILE>. If this option is not selected, the default operation is to append the copied information to the end of an existing file with the same name that you specify.

noheader — This copies the information into a file without headings.

pod_command — This allows you to copy the most recent commands sent to the HP 64700 Series emulator/analyzer.

printer — This option specifies your system printer as the destination device for the **copy** command. Before you can specify the printer as the destination device, you must define PRINTER as a shell variable. For example, you could enter the text shown below after the "$" symbol:

```
$ PRINTER=lp
$ export PRINTER
```

If you don't want the print message to overwrite the command line, execute:

```
$ set PRINTER = "lp -s"
```

registers — This allows you to copy a list of the contents of the emulation processor registers to the selected destination. The **copy register** command is not allowed when the emulator is configured for real-time runs and is running the user program. With no options specified, the basic register class is copied.

<REGISTER> — Specifies the name of an individual register. Refer to the "Accessing Registers" section in the "Using the Emulator" chapter for a list of the register names and classes.

software _breakpoints — This option lets you copy a list of the current software breakpoints to a selected destination.

status — This allows you to copy emulation and analysis status information.

--SYMB-- — This option represents the symbol whose children are to be listed. See the **--SYMB--** syntax diagram and the *Symbolic Retrieval Utilities User's Guide* for information on symbol hierarchy.

thru_line_number — Specifies the last line number of the trace list to include in the copied range.

to — This allows you to specify a destination for the copied information.

trace — This lets you copy the current trace listing to the selected destination. Trace information is copied in the same format as specified in the last display trace command. Initial values are the same as specified by the last **display trace** command.

Examples

copy local_symbols_in mod_name ***to printer*** <RETURN>

copy local_symbols_in mod_name: ***to*** linenumfile <RETURN>

copy memory start ***to printer*** <RETURN>

copy memory 0 ***thru*** 100h , start ***thru*** +5 , 500H ,
target2 ***to*** memlist <RETURN>

copy memory 2000h ***thru*** 204fh ***to*** memlist <RETURN>

copy registers BASIC ***to printer*** <RETURN>

copy registers to reglist <RETURN>

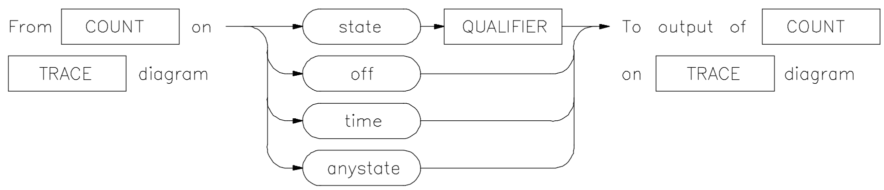
copy trace to tlist <RETURN>

copy trace from_line_number 0 ***thru_line_number*** 5
to longtrac <RETURN>

See Also

The **display**, **modify**, and **store** commands.

COUNT



The analyzer trace depth is 1024 states. A state is a unique combination of address, data, and status values occurring on the emulation bus simultaneously. When counting is off, the analyzer can store 1024 states in the trace buffer. When counting is on, the analyzer can only store 512 states in the trace buffer. That is because the analyzer must now use two states in the trace buffer for each state captured on the analyzer bus. One of the two states stores the state information itself. The other state stores the count information associated with the state information.

By default, all stored states are displayed. You can decrease the depth of the trace display buffer to speed the display of the trace. The following command decreases the trace display depth to 256 states:

display trace depth 256 <RETURN>

By default, the analyzer counting function is turned off.

anystate — This option allows you to set up the **counting** parameter for the analyzer to count on any state.

off — This option turns off trace counting capability. As previously explained, turning off counting provides a larger trace depth.

QUALIFIER — This is defined by you and used with the **state** option to define the states to be captured by the analyzer.

state — This causes the emulation-bus analyzer to count occurrences of the specified state during a trace measurement.

time — This option causes the emulation-bus analyzer to count the time between states captured during the trace measurement.

Examples

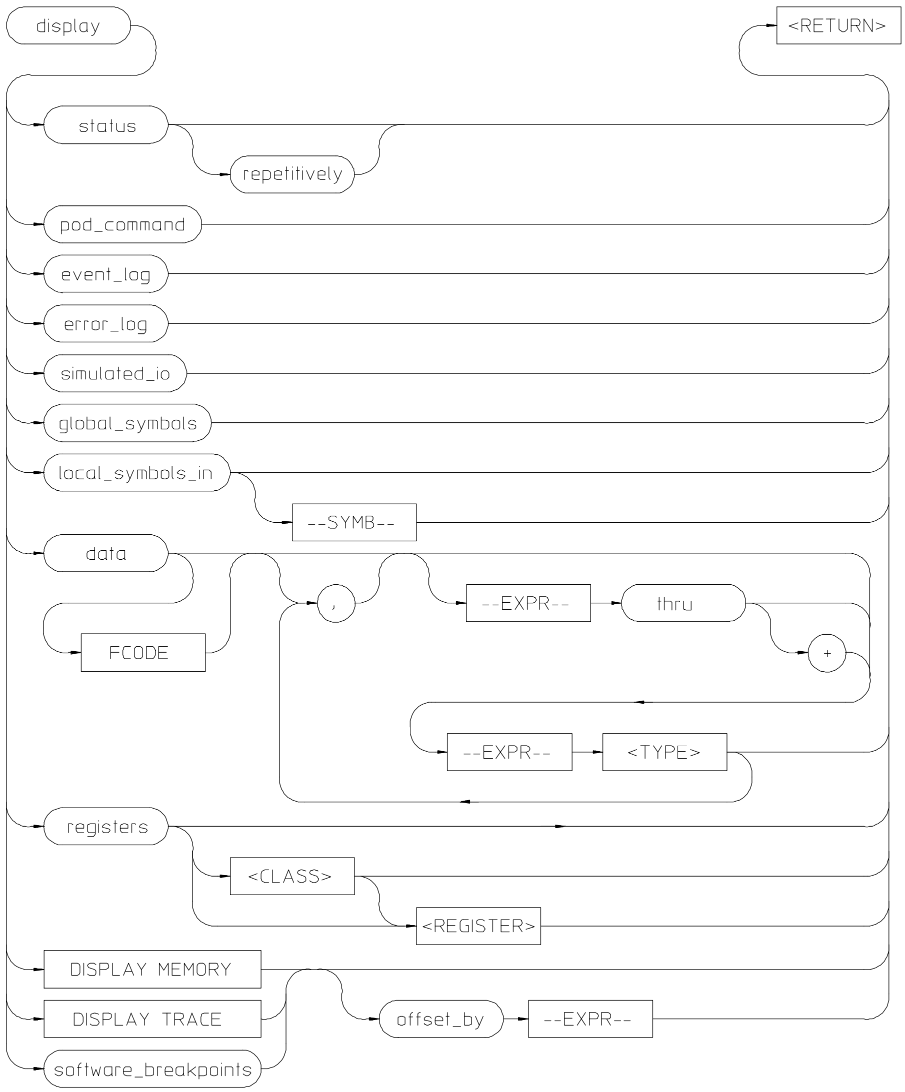
`**trace after** START **counting state** LOOP2 <RETURN>`

`**trace counting time** <RETURN>`

See Also

The **trace** command.

display



This command displays selected information on your screen.

You can use the <Up arrow>, <Down arrow>, <PREV>, and <NEXT> keys to view the displayed information. For software_breakpoints, data, memory, and trace displays you can use the <CTRL>g and <CTRL>f keys to scroll left and right if the information goes past the edge of the screen.

Depending on the information you select, defaults may be the options selected for the previous execution of the **display** command.

The parameters are as follows:

, A leading comma allows you to append additional expressions to the previous display data command. Commas between expression/data type specifications allow you to specify multiple variables and types for display with the current command.

<CLASS> Specifies a particular class of the emulator registers. Refer to the "Accessing Registers" section in the "Using the Emulator" chapter for a list of the register names and classes.

data You can display the values of simple data types in your program. This command can save you time; otherwise, you would need to search through memory displays for the location and value of a particular variable.

The address, identifier, and data value of each symbol may be displayed. You must issue the command **set symbols on** to see the symbol names displayed.

In the first display data command after you begin an emulation session, you must supply at least one expression specifying the data item(s) to display.

Thereafter, the display data command defaults to the expressions specified in the last display data command, unless new expressions are supplied or appended (with a leading comma).

Symbols are normally set off until you give the command **set symbols on**. Otherwise, only the address, data type, and value of the data item will be displayed.

error_log This option displays the recorded list of error messages that occurred during the emulation session.

event_log This option displays the recorded list of events.

--EXPR-- An expression is a combination of numeric values, symbols, operators, and parentheses, specifying the data item to display or an offset value for the breakpoint address. See the **--EXPR--** syntax diagram.

global_symbols — This command displays the global symbols defined for the current absolute file. Global symbols are symbols declared as global in the source file. They include procedure names, variables, constants, and file names. When the **display global_symbols** command is used, the listing will include the symbol name and its logical address.

local_symbols_in — This option lets you display all the children of a given symbol. Displaying the local symbols sets the current working symbol to the one specified. See the **--SYMB--** syntax page and the *Symbolic Retrieval Utilities User's Guide* for details on symbol hierarchy.

memory — This option allows you to display the contents of memory.

offset_by — This option allows you to offset the listed software breakpoint address value from the actual address of the breakpoint. By subtracting the offset value from the breakpoint address, the system can cause the listed address to match that given in the assembler or compiler listing or symbols after code has been relocated.

pod_command — This option lets you display the output of previously executed emulator pod commands.

registers — This command displays the current contents of the emulation processor registers.

If a **step** command just executed, the mnemonic representation of the last instruction is also displayed, if the current display is the register display. The **display register** command is not allowed when the emulator is configured for real-time runs and is running the user program. Symbols also may be displayed in the register step mnemonic string (see **set symbols**). With no options specified, the basic register class is displayed as the default.

<REGISTER> — Specifies the name of an individual register. Refer to the "Accessing Registers" section in the "Using the Emulator" chapter for a list of the register names and classes.

repetitively — This optional part of the **display status** command causes the emulator status screen to be updated continuously.

simulated_io — This lets you display data written to the simulated I/O display buffer after you have enabled polling for simulated I/O in the emulation configuration. For details about setting up simulated I/O, refer to the *Simulated I/O User's Guide.*

software_ breakpoints — This option lets you display the current list of software breakpoints.

If the emulation session is continued from a previous session, the listing will include any previously defined breakpoints. The column marked "status" shows whether the breakpoint is pending, inactivated, or unknown.

A pending breakpoint causes the processor to enter the emulation monitor upon execution of that breakpoint. Executed breakpoints are listed as inactivated. Entries that show an inactive status can be reactivated by executing the **modify software_breakpoints set** command.

A label column also may be displayed for addresses that correspond to a symbol. See the **set** command for details.

status — This displays the emulator and trace status.

--SYMB-- — This option represents the symbol whose children are to be listed. See the **--SYMB--** syntax diagram and the *Symbolic Retrieval Utilities User's Guide* for more information on symbol hierarchy and representation.

thru --EXPR-- — Allows you to specify a range of addresses for which you want data display. Typically, you use this to display the contents of an array. You can display both single-dimensioned and multi-dimensioned arrays. Arrays are displayed in the order specified by the language definition, typically row major order for most Algol-like languages.

trace — This displays the current trace list.

<TYPE> — Specifies the format in which to display the information, as shown in the following table. (Data type information is not available from the symbol database, so you must specify it.)

<TYPE>	Meaning
byte	Hex display of one 8-bit location.
word	Hex display of one 16-bit location.
long	Hex display of one 32-bit location.
int8	Display of one 8-bit location as a signed integer using two's complement notation.
int16	Display of two bytes as a signed integer using two's complement notation.
int32	Display of four bytes as a signed integer using two's complement notation.
u_int8	Display of one byte as an unsigned positive integer.
u_int16	Display of two bytes as an unsigned positive integer.
u_int32	Display of four bytes as an unsigned positive integer.
char	Displays one byte as an ASCII character in the range 0..127. Control characters and values in the range 128..255 are displayed as a period (.).

Examples

```
display event_log <RETURN>

display local_symbols_in mod_name <RETURN>

display data Msg_A thru +17 char, Stack long <RETURN>

set symbols on <RETURN>

set width label 30 <RETURN>

display data , Msg_B thru +17 char, Msg_Dest thru +17
char <RETURN>

display local_symbols_in  mod_name <RETURN>

display local_symbols_in mod_name:main <RETURN>
```

display registers <RETURN>

display registers BASIC D2 <RETURN>

display simulated_io <RETURN>

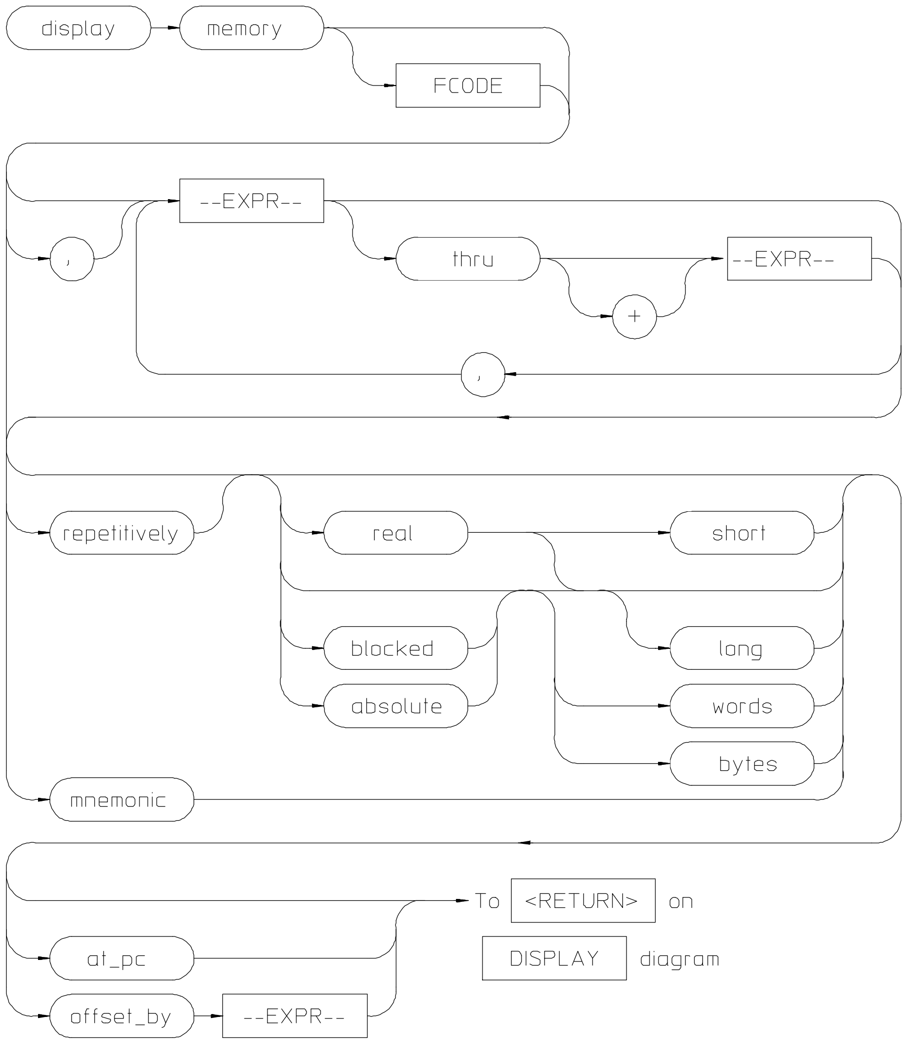
display software_breakpoints <RETURN>

display software_breakpoints offset_by 1000H <RETURN>

See Also

The **copy**, **modify**, **set**, and **step** command descriptions and the following pages which describe the **display memory** and **display trace** commands.

display memory



This command displays the contents of the specified memory location or series of locations.

The memory contents can be displayed in mnemonic, hexadecimal, or real number format. In addition, the memory addresses can be listed offset by a value, which allows the information to be easily compared to the program listing.

When displaying memory mnemonic and stepping, the next instruction that will step is highlighted. The memory mnemonic display autopages to the new address if the next PC goes outside the currently displayed address range. This feature works even if stepping is performed in a different emulation window than the one displaying memory mnemonic.

Pending software breakpoints are shown in the memory mnemonic display by an asterisk (*) in the leftmost column of the assembly instruction or source line that has a pending breakpoint.

A label column (symbols) may be displayed for all memory displays except blocked mode. Memory mnemonic may be displayed with source and assembly code intermixed, or with source code only. Symbols also can be displayed in the memory mnemonic string. (See the **set** command.)

Initial values are the same as specified by the command:

display memory 0 ***blocked bytes offset_by*** 0

Defaults are values specified in a previous **display memory** command.

The symbols and source defaults are:

set source off symbols off

The parameters are as follows:

absolute — Formats the memory listing in a single column.

at_pc — Displays the memory at the address pointed to by the current program counter value.

blocked — Formats the memory listing in multiple columns.

bytes — Displays the absolute or blocked memory listing as byte values.

--EXPR-- — An expression is a combination of numeric values, symbols, operators, and parentheses, specifying a memory address or memory offset value. See the EXPR syntax diagram.

FCODE — The function code used to define the address space being referenced. See the syntax diagram for FCODE to see a list of the function codes available and for an explanation of those codes.

long — Displays memory in a 64-bit real number format or 32-bit long words when preceded by **blocked** or **absolute**.

mnemonic — This causes the memory listing to be formatted in assembly language instruction mnemonics with associated operands. When specifying mnemonic format, you should include a starting address that corresponds to the first byte of an operand to ensure that the listed mnemonics are correct. If **set source only** is on, you will see only the high level language statements and corresponding line numbers.

offset_by — This option lets you specify an offset that is subtracted from each of the absolute addresses before the addresses and corresponding memory contents are listed. You might select the offset value so that each module appears to start at address 0000H. The memory contents listing will then appear similar to the assembler or compiler listing.

This option is also useful for displaying symbols and source lines in dynamically relocated programs.

previous_display — Returns to display associated with the previous mnemonic memory display command.

real — Formats memory values in the listing as real numbers. (NaN in the display list means "Not a Number.")

repetitively — Updates the memory listing display continuously. You should only use this to monitor memory while running user code, since it is very CPU intensive. To allow updates to the current memory display whenever memory is modified, a file is loaded, software breakpoint is set, etc., use the **set update** command.

short — Formats the memory list as 32-bit real numbers.

thru — This option lets you specify a range of memory locations to be displayed. Use the <Up arrow>, <Down arrow>, <NEXT>, and <PREV> keys to view additional memory locations.

words — Displays the absolute or blocked memory listing as 16-bit word values.

, — A comma after **memory** in the command line appends the current **display memory** command to the preceding **display memory** command. The data specified in both commands is displayed. The data will be formatted as specified in the current

command. The comma is also a delimiter between values when specifying multiple addresses.

Examples

You can display memory in real number and mnemonic formats:

display memory 2000h ***thru*** 202fh , 2100h ***real long*** <RETURN>

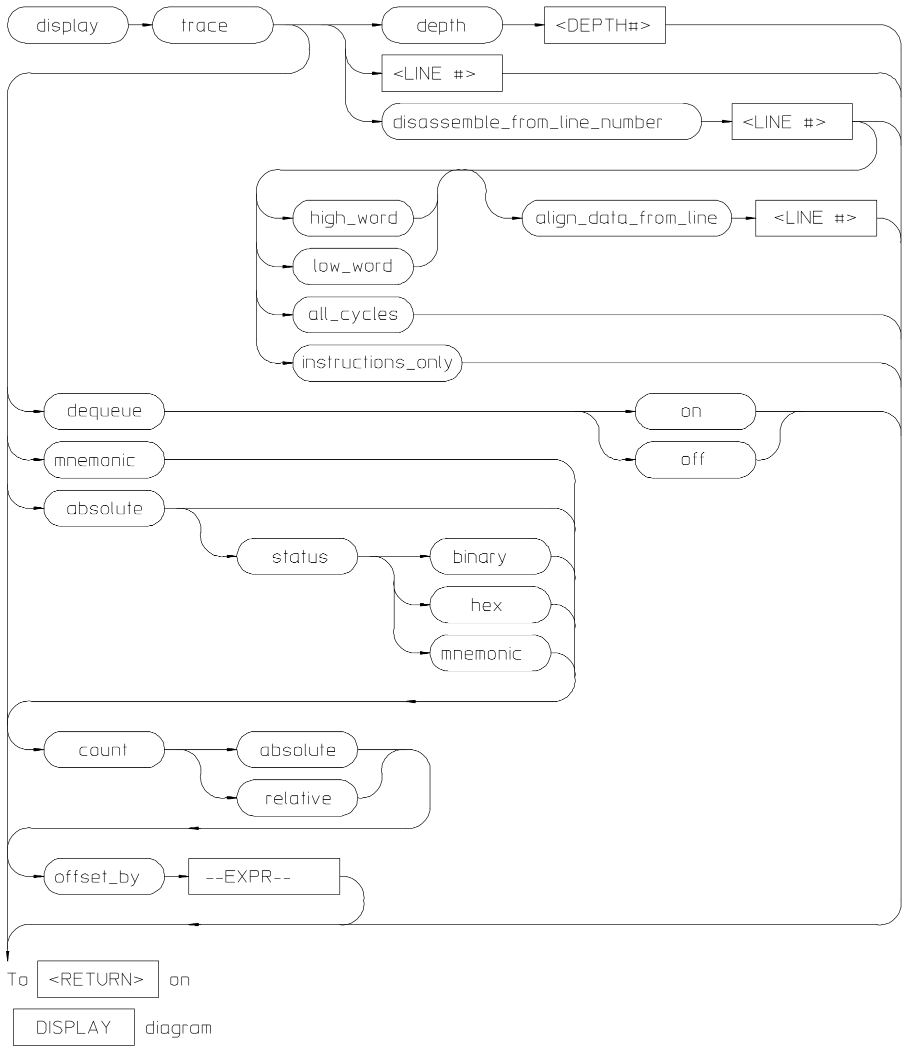
display memory 400h ***mnemonic*** <RETURN>

set symbols on <RETURN>
set source on <RETURN>
display memory main mnemonic <RETURN>

See Also

The **copy memory**, **modify memory**, **set**, and **store memory** commands.

display trace



This command displays the contents of the trace buffer.

Captured information can be presented as absolute hexadecimal values or in mnemonic form. The processor status values captured by the analyzer can be listed mnemonically or in hexadecimal or binary form.

Addresses captured by the analyzer are physical addresses.

The **offset_by** option subtracts the specified offset from the addresses of the executed instructions before listing the trace. With an appropriate entry for **offset**, each instruction in the listed trace will appear as it does in the assembled or compiled program listing.

The **count** parameter lists the current trace of time or state either relative to the previous event in the trace list or as an absolute count measured from the trigger event. If time counts are currently selected, the **count** parameter causes an absolute or relative time count to be listed. If the current trace contains state counts, a relative or absolute state count results.

The **source** parameter allows display of source program lines in the trace listing, enabling you to quickly correlate the trace list with your source program.

Initial values are the same as specified by the command:

display trace mnemonic count relative offset_by 0
<RETURN>

The parameters are as follows:

absolute — Lists trace information in hexadecimal format, rather than mnemonic opcodes.

align_data_ from_line — Use this to correct data-alignment problems if you see any in a dequeued trace list. If you see that the dequeuer has aligned data with the wrong instructions, use this token to select the correct data alignment by specifying the line that should begin a data realignment (align_data_from_line 36).

all_cycles — Used to specify that all cycles should be included in the inverse assembled information shown in the trace list.

count

absolute — This lists the time count for each event of the trace as the total time measured from the trigger event.

relative — This lists the time count for each event of the trace as the time measured relative to the previous event.

depth

<DEPTH#>

This defines the number of states to be uploaded by the interface.

Note that after you have changed the trace depth, execute the command **wait measurement_complete** before displaying the trace. Otherwise the new trace states will not be available.

dequeue

This obtains a trace list showing the activity of the emulation processor during the trace. Unused prefetches are eliminated from this display, and data transactions are aligned with the instructions that caused them to occur.

disassemble_from_line_number

Displays the trace at a certain line number and disassembles instruction opcodes.

--EXPR--

An expression is a combination of numeric values, symbols, operators, and parentheses, specifying an offset value to be subtracted from the addresses traced by the emulation analyzer. See the EXPR syntax diagram.

high_word

Causes inverse assembly to begin with the opcode in the high word of the long word located in the specified trace-memory line number. This field is isn't applicable for the 68340 because its data bus is 16-bits wide.

instructions_only

Causes the trace list to contain only those lines that show an instruction opcode.

<LINE#>

This prompts you for the trace list line number to be centered in the display. Also, you can use <LINE#> with **disassemble_from_line_number**. <LINE#> prompts you for the line number from which the inverse assembler attempts to disassemble data in the trace list. When used with **align_data_from_line**, this is the line number from which to begin aligning data. The line number specified for data alignment must be the same as, or higher than, the line number specified for the beginning of the trace disassembly.

low_word

Causes inverse assembly to begin with the opcode stored in the low word of the long word at the specified trace memory line number. This field is isn't applicable for the 68340 because its data bus is 16-bits wide.

mnemonic

Lists trace information with opcodes in mnemonic format.

offset_by

This option allows you to offset the listed address value from the address of the instruction. By subtracting the offset value from the physical address of the instruction, the system makes the listed address match that given in the assembler or compiler listing.

This option is also useful for displaying symbols and source lines in dynamically relocated programs.

Note that when using the **set source only** command, the analyzer may operate more slowly than when using the **set source on** command. This is an operating characteristic of the analyzer:

> When you use the command **set source on**, and are executing only assembly language code (not high-level language code), no source lines are displayed. The trace list will then fill immediately with the captured assembly language instructions.
>
> When using **set source only**, no inverse assembled code is displayed. Therefore, the emulation software will try to fill the display with high-level source code. This requires the emulation software to search for any captured analysis data generated by a high-level language statement.

In conclusion, you should not set the trace list to **set source only** when tracing assembly code. This will result in optimum analyzer performance.

status

binary — Lists absolute status information in binary form.

hex — Lists absolute status information in hexadecimal form.

mnemonic — Lists absolute status information in mnemonic form.

Examples

display trace count absolute <RETURN>

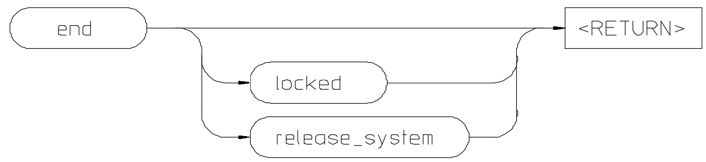
display trace absolute status binary <RETURN>

display trace mnemonic <RETURN>

See Also

The **copy**, **store**, and **set** commands.

end



This command terminates the current emulation session.

You can end the emulation session and keep the emulator in a locked state. The current emulation configuration is stored, so that you can continue the emulation session on reentry to the emulator. You can select another measurement system when ending the current session. You also can release the emulation system when ending the session so that others may use the emulator.

Note that pressing <CTRL>d performs the same operation as pressing **end <RETURN>**. Pressing <CTRL>\ or <CTRL>| performs the same as **end release_system <RETURN>**.

If the command is specified without any options, two different things can occur. What occurs depends on whether the interface instance being ended is the only one currently executing. If it is, then this command ends the interface, but leaves the emulator in a continue state. The interface, if restarted, will reload the last configuration used.

If the interface instance is one of several into the same emulation session, then the end command ends the interface instance where the end is issued. Other interface instances into the same emulation session are not affected.

The parameters are as follows:

locked — This option allows you to stop all active instances of an emulator/analyzer interface session in one or more windows and/or terminals. When the emulation session ends, control returns to the UNIX shell, but the emulator is still locked to your user id and is not available to others.

release_system — This option stops all instances of the emulator/analyzer interface in one or more windows or terminals. The emulation system is released for other users. If you do not release the emulation system, others cannot access it.

Examples

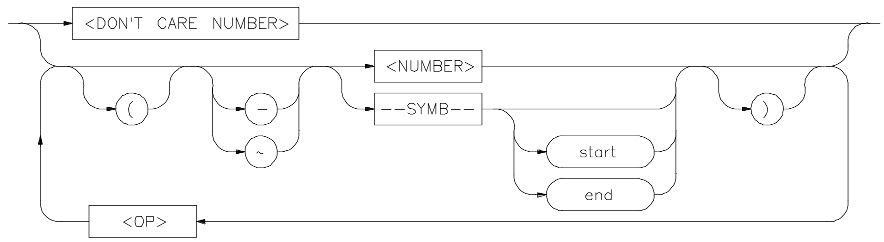
end <RETURN>

end release_system <RETURN>

See Also

The "Exiting the Emulator/Analyzer Interface" section in the "Using the Emulator/Analyzer Interface" chapter.

--EXPR--



An expression is a combination of numeric values, symbols, operators, and parentheses used to specify address, data, status, executed address, or any other value used in the emulation commands.

The function of an expression (**--**EXPR**--**) is to let you define the address, data, status, or executed address expression that fits your needs. You can combine multiple values to define the expression.

Certain emulation commands will allow the option of <+EXPR> after pressing a thru softkey. This allows you to enter a range without retyping the original base address or symbol. For example, you could specify the address range

`disp_buf` ***`thru`*** `disp_buf + 25`

as

`disp_buf` ***`thru`*** `+25`

The parameters are as follows:

DON'T CARE NUMBER You can include "don't care numbers" in expressions. These are indicated by a number containing an "x." These numbers may be defined as binary, octal, decimal, or hexadecimal. For example: 1fxxh, 17x7o, and 011xxx10b are valid.

Note that "Don't care numbers" are not valid for all commands.

--NORMAL-- This appears as a softkey label to enable you to return to the **--EXPR--** key. The **--NORMAL--** label can be accessed whenever defining an expression, but is only valid when "C" appears on the status line, which indicates a valid expression has been defined.

<NUMBER>

This can be an integer in any base (binary, octal, decimal, or hexadecimal), or can be a string of characters enclosed with quotation marks.

<OP>

This represents an algebraic or logical operand and may be any of the following (in order of precedence):

mod	modulo
*	multiplication
/	division
&	logical AND
+	addition
-	subtraction
|	logical OR

--SYMB--

This allows you to define symbolic information for an address, range of addresses, or a file. See the **--SYMB--** syntax pages and the *Symbolic Retrieval Utilities User's Guide* for more information on symbols.

end

This displays the last location where the symbol information may be located. For example, if a particular symbol is associated with a range of addresses, **end** will represent the last address in that range.

start

This displays first memory location where the symbol you specify may be located. For example, if a particular symbol is associated with a range of addresses, **start** will represent the first address in that range.

<UNARY>

This defines either the algebraic negation (minus) sign (**-**) or the logical negation (NOT) sign (~).

( )

Parentheses may be used in expressions to enclose numbers. For every opening parenthesis, a closing parenthesis must exist.

Note that when "C" appears on the right side of the status line, a valid expression exists. The **--NORMAL--** key can be accessed at any time, but is only valid when "C" is on the command line.

Note that when a **thru** softkey has been entered, a <+ EXPR> prompt appears. This saves you from tedious repeated entry of long symbols and expressions. For example:

```
disp_buf thru +25
```

is the same as

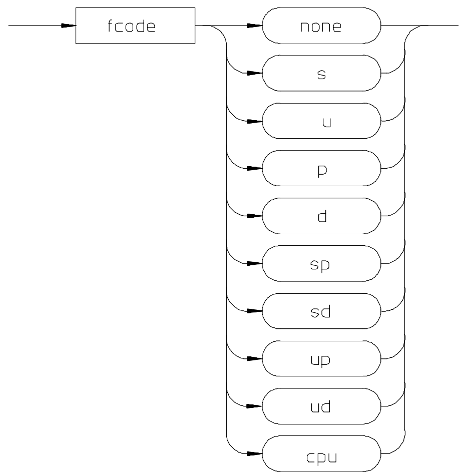
```
disp_buf thru disp_buf + 25
```

Examples

```
05fxh

0ffffh

disp_buf + 5

symb_tbl + (offset / 2)

start

mod_name: line 15 end
```

See Also The SYMB syntax description.

FCODE



The function code is used to define the address space being referenced. Select the appropriate function code from those listed below.

cpu	CPU space.
d	Data space.
none	Causes the emulator to ignore the function code bits.
p	Program space.
s	Supervisor space.
sd	Supervisor data space.
sp	Supervisor program space.
u	User space.
ud	User data space.
up	User program space.

Examples

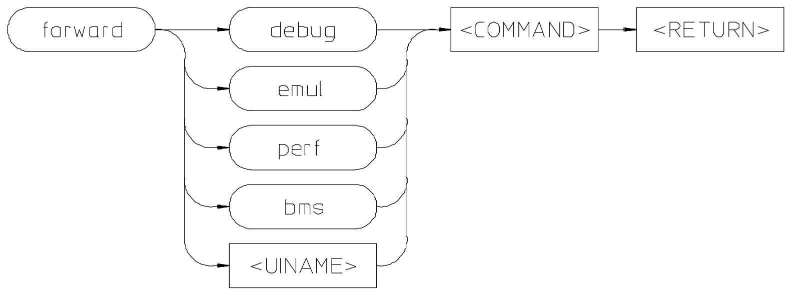
To copy a portion of user data memory to a file:

copy memory fcode ud 1000H **thru** 1fffH **to** mymem <RETURN>

To modify a location in program memory:

modify memory fcode p 5000h **long to** 12345678h <RETURN>

forward



This command lets you forward commands to other HP 64700 interfaces that use the "emul700dmn" daemon process to coordinate actions between the interfaces.

bms — Sends messages to the Broadcast Message Server or BMS.

<COMMAND> — An ASCII string, enclosed in quotes, that is the command to be forwarded to the named interface.

debug — Forwards command to the high-level debugger interface.

emul — Forwards command to the emulator/analyzer interface.

perf — Forwards commands to the software performance analyzer interface.

<UINAME> — Forwards commands to a user interface name other than those available on the softkeys.

Examples

To send the "Program Run" command to the debugger:

`forward debug` `"Program Run" <RETURN>`

To send the "profile" command to the software performance analyzer:

`forward debug` `"profile" <RETURN>`

See Also

The *User's Guide* for the interface to which you are forwarding commands.

help

Displays information about system and emulation features during an emulation session.

Typing **help** or **?** displays softkey labels that list the options on which you may receive help. When you select an option, the system will list the information to the screen.

The **help** command is not displayed on the softkeys. You must enter it into the keyboard. You may use a question mark in place of **help** to access the help information.

The parameters are as follows:

<HELP_FILE> This represents one of the available options on the softkey labels. You can either press a softkey representing the help file, or type in the help file name. If you are typing in the help file name, make sure you use the complete syntax. Not all of the softkey labels reflect the complete file name.

Examples

help ***system_commands*** <RETURN>

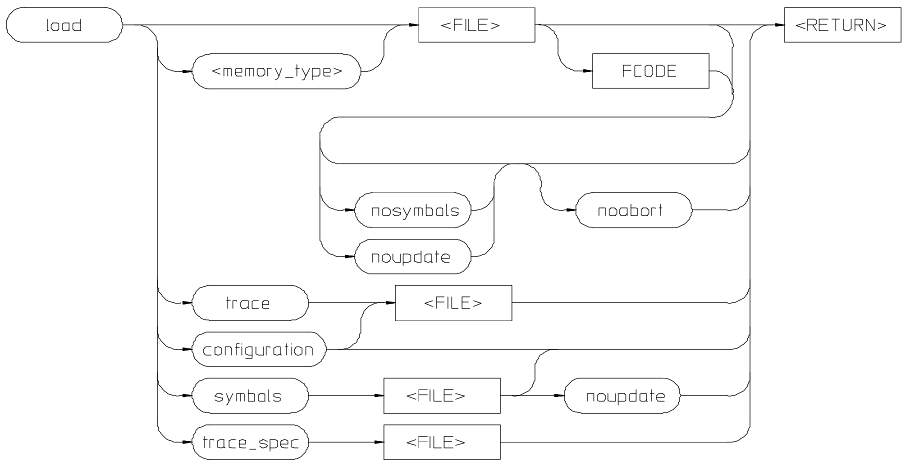
? ***run*** <RETURN>

This is a summary of the commands that appear on the softkey labels when you type **help** or press **?**:

system_commands
run
trace
step
break
display
modify
load
store

copy
reset
stop_trace
end
software_breakpoints
registers
expressions (--EXPR--)
symbols (--SYMB--)
specify
cmb
cmb_execute
map
set
wait
pod_command
bbaunload
coverage
performance_measurement_initialize
performance_measurement_run
performance_measurement_end

load



This command transfers absolute files from the host computer into emulation or target system RAM. With other parameters, the load command can load emulator configuration files, trace records, trace specifications, or symbol files.

The absolute file contains information about where the file is stored. The memory map specifies that the locations of the file are in user (target system) memory or emulation memory. This command also allows you to access and display previously stored trace data, load a previously created configuration file, and load absolute files with symbols.

Note that any file specified by <FILE> cannot be named "configuration", "emul_mem", "user_mem", "symbols", "trace", or "trace_spec" because these are reserved words, and are not recognized by the emulator/analyzer interface as ordinary file names.

The absolute file is loaded into emulation memory by default.

The parameters are as follows:

configuration

This option specifies that a previously created emulation configuration file will be loaded into the emulator. You can follow this option with a file name. Otherwise the previously loaded configuration will be reloaded.

FCODE — The function code used to define the address space being referenced. See the syntax diagram for FCODE to see a list of the function codes available and for an explanation of those codes.

<FILE> — This represents the absolute file to be loaded into either target system memory, emulation memory (.X files are assumed), or the trace memory (.TR files are assumed).

<memory_type> — This indicates the type of memory that you choose for the load operation. The memory type can be emulation or user memory. You also can load a background monitor file.

noabort — This option allows you to load a file even if part of the file is located at memory mapped as "guarded" or "target ROM" (trom).

nosymbols — This option causes the file specified to be loaded without symbols.

noupdate — This option suppresses rebuilding of the symbol data base when you load an absolute file. If you load an absolute file, end emulation, then modify the file (and relink it), the symbol database will not be updated upon reentering emulation and reloading the file. The default is to rebuild the database.

symbols — This option causes the file specified to be loaded with symbols.

trace — This option allows you to load a previously generated trace file.

trace_spec — This option allows you to load a previously generated trace specification.

Note that the current trace specification will be modified, but a new trace will not be started. To start a trace with the newly loaded trace specification, enter **trace again** or **specify trace again** (not **trace**). If you specify **trace**, a new trace will begin with the default trace specification, not the one you loaded.

Examples

load sort1 <RETURN>

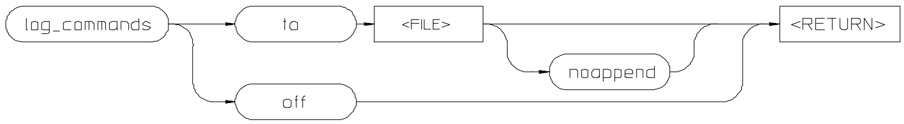
load configuration config3 <RETURN>

load trace trace3 <RETURN>

See Also

The **display trace** command.

log_commands



This command allows you to record commands that are executed during an emulation session.

A command file is an ASCII file containing Softkey Interface commands. The interface can read a command file and execute its commands as if they were typed into the command line. Simply type the filename on the command line.

This interface command lets you create command files by logging. When the interface is in logging mode, all commands entered and executed on the command line are also copied to the named file. Once started, logging continues until either logging is turned off or the emulation session is ended.

The **log_commands** command is not on the softkeys. You must type it into the command line to access the remainder of the log_commands softkeys.

The parameters are as follows:

<FILE> — This represents the file where you want to store interface commands. If the file does not exist, a new file is created. If the file already exists, the new commands are appended to the present content in the file, unless the **noappend** option is specified.

off — This option stops command logging.

noappend — If the named file is an existing file, this option causes the new commands to overwrite any information present in the file. If this option is not specified, new commands are appended to the existing contents of the file.

Examples

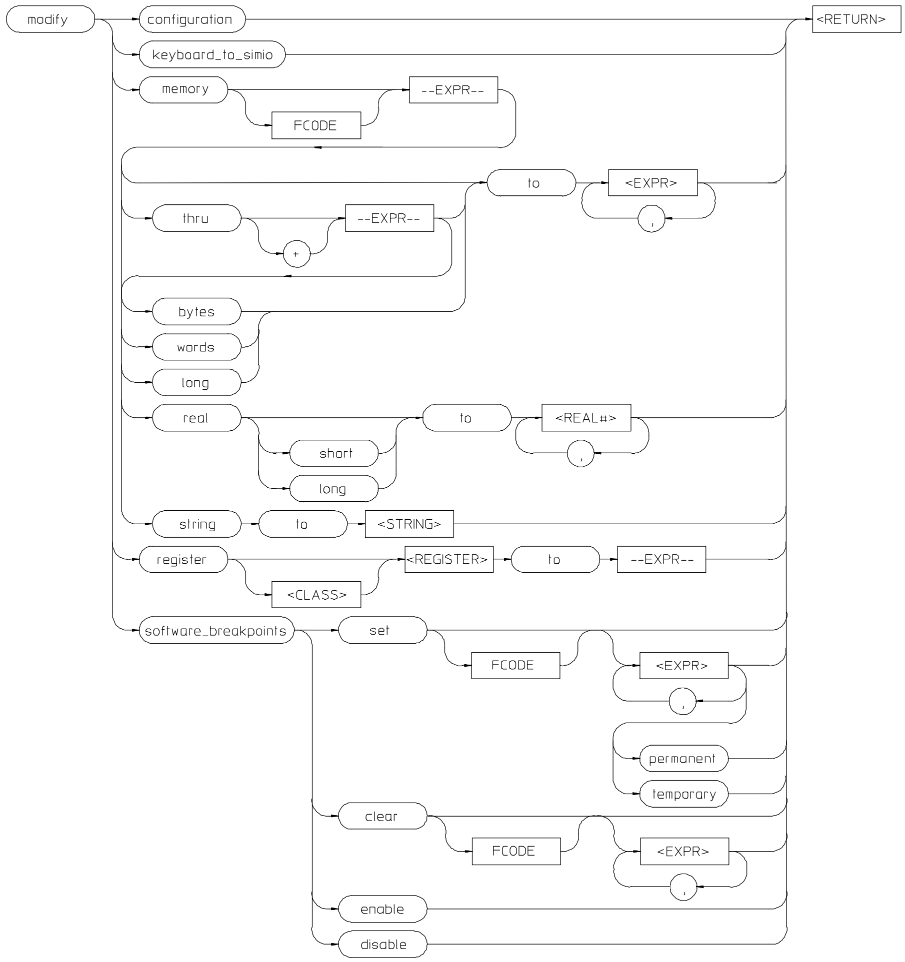
log_commands ***to*** logfile <RETURN>

log_commands ***off*** <RETURN>

See Also

The **wait** command.

modify



This command allows you to observe or change information specific to the emulator.

The **modify** command is used to:

- Modify contents of memory (as integers, strings, or real numbers).
- Modify the contents of the processor registers.
- View or edit the current emulation configuration.
- Modify the software breakpoints table.

The following pages contain detailed information about the various **modify** syntax diagrams.

The parameters are as follows:

bytes — Modify memory in byte values.

<CLASS> — Specifies a particular class of the emulator registers. Refer to the "Accessing Registers" section in the "Using the Emulator" chapter for a list of the register names and classes.

clear — This option erases the specified breakpoint address and restores the original content of the memory location. (The location must not have changed (by loading a file or modifying memory) after the breakpoint was set.) If no breakpoints are specified in the command, all currently specified breakpoints are cleared and the memory locations are restored to their original values.

, — A comma is used as a delimiter between values when modifying multiple memory addresses or breakpoints.

configuration — Lets you view and edit the current emulation configuration items. The configuration questions are presented in sequence with either the default response, or the previously entered response. You can select the currently displayed response by pressing <RETURN>. Otherwise, you can modify the response as you desire, then press <RETURN>.

disable — This option turns off the software breakpoint capability.

enable — This option allows you to modify the software breakpoint specification.

--EXPR-- — An expression is a combination of numeric values, symbols, operators, and parentheses, specifying a memory address, register, or software breakpoint value. See the **--EXPR--** description.

FCODE — The function code used to define the address space being referenced. See the syntax diagram for FCODE to see a list of the function codes available and for an explanation of those codes.

keyboard_to_simio — Lets the keyboard to interact with your program through the simulated I/O software. When the keyboard is activated for simulated I/O, its normal interaction with emulation is disabled. The emulation softkeys are blank and the softkey labeled "suspend" is displayed on your screen. Pressing **suspend <RETURN>** will deactivate keyboard simulated I/O and return the keyboard to normal emulation mode. For details about setting up simulated I/O, refer to the *Simulated I/O User's Guide*.

long — Modify memory values as 32-bit long word values or 64-bit real values when preceded by **real**.

memory — Lets you modify the contents of selected memory locations. You can **modify** the contents of individual memory locations to individual values. Or, you can modify a range of memory to a single value or a sequence of values.

Modify a series of memory locations by specifying the address of the first location in the series to be modified, and the values to which the contents of that location and successive locations are to be changed. The first value listed will replace the contents of the first memory location. The second value replaces the contents of the next memory location in the series, and so on, until the list is exhausted. When more than one value is listed, the value representations must be separated by commas. (See the examples for more information.)

A range of memory can be modified such that the content of each location in the range is changed to the single specified value, or to a single or repeated sequence. This type of memory modification is done by entering the limits of the memory range to be modified (--EXPR-- thru --EXPR--) and the value or list of values (--EXPR--, ... , --EXPR--) to which the contents of all locations in the range are to be changed.

Note that if the specified address range is not large enough to contain the new data, only the specified addresses are modified.

If the address range contains an odd number of bytes and a word operation is being executed, the last word of the address range will be modified. Thus the memory modification will stop one byte after the end of the specified address range.

If an error occurs in writing to memory (to guarded memory or target memory with no monitor) the modification is aborted at the address where the error occurred.

Memory modifications of integer values use the currently set display memory mode. Byte is the default.

Memory modifications of "real" values use the currently set display memory mode. Short is the default.

permanent — Sets a permanent breakpoint. The software breakpoint instruction remains in the program until the breakpoint is inactivated or removed.

real — Modify memory as real number values.

<REAL#> — This prompts you to enter a real number as the value.

register — Lets you to modify the contents of the emulation processor internal registers. The **modify register** command is not allowed when the emulator is configured for real-time runs and is running the user program.

<REGISTER> — Specifies the name of an individual register. Refer to the "Accessing Registers" section in the "Using the Emulator" chapter for a list of the register names and classes.

set — This option allows you to activate software breakpoints in your program. If no breakpoint addresses are specified in the command, all breakpoints that have been inactivated (executed) are reactivated.

short — Modify memory values as 32-bit real numbers.

software_ breakpoints — Lets you change the specification of software breakpoints. Software breakpoints provide a way to accurately stop the execution of your program at one or more instruction locations. When a software breakpoint is set, the instruction that is normally at that location is replaced with a BGND instruction. When the software breakpoint is executed, control is passed to the emulator's monitor program, and the original instruction is restored in the user program. Thus, execution is interrupted before the instruction at the specified address is executed.

Operation of the program can be resumed after the breakpoint is encountered, by specifying either a **run** or **step** command.

If you modify software breakpoints while the memory mnemonic display is active, the new breakpoints are indicated by a "*****" in the leftmost column of the instruction containing the breakpoint.

The software breakpoint facility may be completely disabled or enabled via the "modify software_breakpoints" command. The default is "enabled".

string — Modify memory values to the ASCII character string given by <STRING>.

<STRING> — Quoted ASCII string including special characters as follows:

null	\0
newline	\n
horizontal tab	\t
backspace	\b
carriage return	\r
form feed	\f
backslash	\\
single quote	\’
bit pattern	\ooo (where ooo is an octal number)

temporary — Sets a temporary breakpoint. When the break occurs, the original opcode is replaced in the program.

thru — This option lets you specify a range of memory locations to be modified.

to — This lets you specify values to which the selected memory locations or registers will be changed.

words — Modify memory locations as 16-bit values.

Examples

modify configuration `<RETURN>`

modify memory `data1` ***bytes to*** `0E3H , 01H , 08H <RETURN>`

modify memory `data1` ***thru*** `DATA100` ***to*** `0FFFFH <RETURN>`

modify memory `0675H` ***real to*** `-1.303 <RETURN>`

modify memory `temp` ***real long to*** `0.5532E-8 <RETURN>`

modify memory `buffer` ***string to*** `"This is a test \n\0" <RETURN>`

modify register `D2` ***to*** `41H <RETURN>`

modify software_breakpoints enable `<RETURN>`

modify software_breakpoints set `loop1 end , loop2 end , 0E40H <RETURN>`

modify software_breakpoints clear `<RETURN>`

modify software_breakpoints set `<RETURN>`

See Also

The **copy**, **display**, **load**, **store** commands.

performance_measurement_end

This command stores data previously generated by the **performance_measurement_run** command, in a file named "perf.out" in the current working directory.

The file named "perf.out" is overwritten each time this command is executed. Current measurement data existing in the emulation system is not altered by this command.

Examples

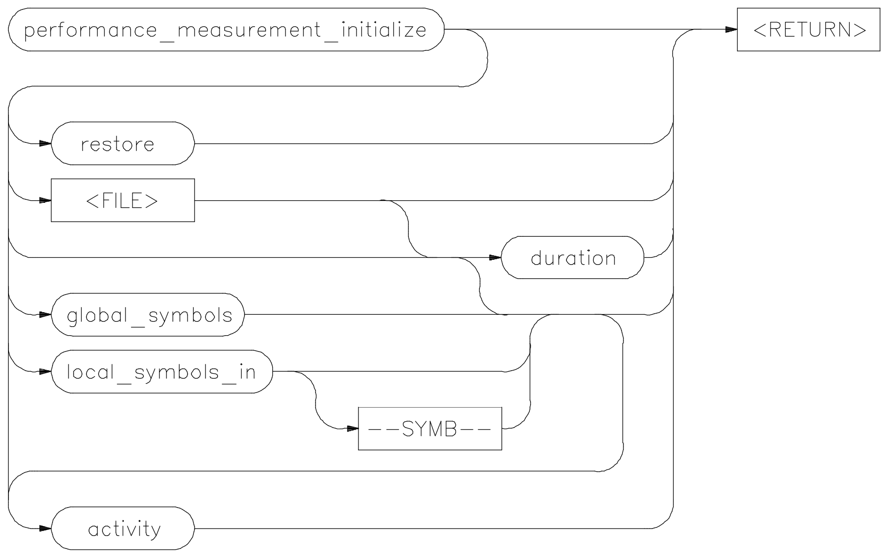
performance_measurement_end <RETURN>

See Also

The **performance_measurement_initialize** and **performance_measurement_run** commands.

Refer to the "Making Software Performance Measurements" chapter for examples of performance measurement specification and use.

performance_measurement_initialize



This command sets up performance measurements.

The emulation system will verify whether a symbolic database has been loaded. If a symbolic database has been loaded, the performance measurement is set up with the addresses of all global procedures and static symbols. If a valid database has not been loaded, the system will default to a predetermined set of addresses, which covers the entire emulation processor address range.

The measurement will default to "activity" mode.

Default values will vary, depending on the type of operation selected, and whether symbols have been loaded.

The parameters are as follows:

activity — This option causes the performance measurement process to operate as though an option is not specified.

duration — This option sets the measurement mode to "duration." Time ranges will default to a predetermined set (unless a user-defined file of time ranges is specified).

<FILE> This represents a file you specify to supply user-defined address or time ranges to the emulator.

global_symbols This option specifies that the performance measurement will be set up with the addresses of all global symbols and procedures in the source program.

local_symbols_in This causes addresses of the local symbols to be used as the default ranges for the measurement.

restore This option restores old measurement data so that a measurement can be continued when using the same **trace** command as previously used.

--SYMB-- This represents the source file that contains the local symbols to be listed. This also can be a program symbol name, in which case all symbols that are local to a function or procedure are used. See the SYMB syntax diagram.

Examples

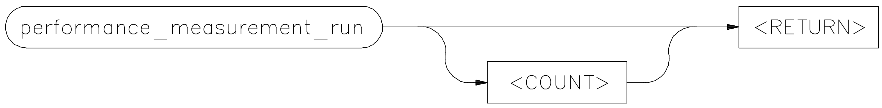
```
performance_measurement_initialize <RETURN>

performance_measurement_initialize duration <RETURN>

performance_measurement_initialize local_symbols_in
mod_name <RETURN>
```

See Also

The **performance_measurement_run** and **performance_measurement_end** commands.

Refer to the "Making Software Performance Measurements" chapter for examples of performance measurement specification and use.

performance_measurement_run



This command begins a performance measurement.

This command causes the emulation system to reduce trace data contained in the emulation analyzer, which will then be used for analysis by the performance measurement software.

The default is to process data presently contained in the analyzer.

The parameters are as follows:

<COUNT>

This represents the number of consecutive traces you specify. The emulation system will execute the trace command, process the resulting data, and combine it with existing data. This sequence will be repeated the number of times specified by the **COUNT** option.

Note that the **trace** command must be set up correctly for the requested measurement. For an activity measurement, you can use the default **trace** command (**trace <RETURN>**).

For a duration measurement, you must set up the trace specification to store only the points of interest. To do this, for example, you could enter:

trace only <symbol_entry> ***or*** <symbol_exit>

Examples

performance_measurement_run 10 <RETURN>

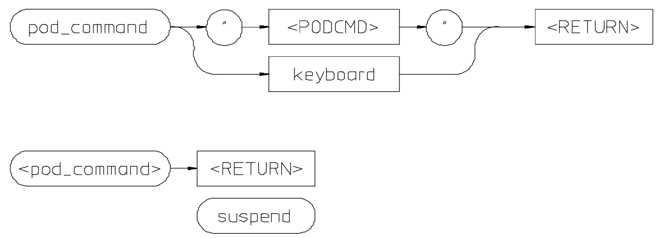
performance_measurement_run <RETURN>

See Also

The **performance_measurement_end** and **performance_measurement_initialize** commands.

Refer to the "Making Software Performance Measurements" chapter for examples of performance measurement specification and use.

pod_command



Allows you to control the emulator through the direct HP 64700 Terminal Interface.

The HP 64700 Card Cage contains a low-level Terminal Interface, which allows you to control the emulator’s functions directly. You can access this interface using **pod_command**. The options to **pod_command** allow you to supply only one command at a time. Or, you can select a keyboard mode which gives you interactive access to the Terminal Interface.

There are certain commands that you should avoid while using the Terminal Interface through **pod_command**.

stty, po, xp	Do not use. These commands will change the operation of the communications channel, and are likely to hang the Softkey Interface and the channel.
echo, mac	Using these may confuse the communications protocols in use on the channel.
wait	Do not use. The pod will enter a wait state, blocking access by the emulator/analyzer interface.
init, pv	These will reset the emulator pod and force an **end release_system** command.
t	Do not use. The trace status polling and unload will become confused.

To see the results of a particular **pod_command** (the information returned by the emulator pod), you use **display pod_command**.

The parameters are as follows:

keyboard — Enters an interactive mode where you can simply type Terminal Interface commands (unquoted) on the command line. Use **display pod_command** to see the results returned from the emulator.

<POD_CMD> — Prompts you for a Terminal Interface command as a quoted string. Enter the command in quotes and press <RETURN>.

suspend — This command is displayed once you have entered keyboard mode. Select it to stop interactive access to the Terminal Interface and return to the Graphical User Interface or Softkey Interface.

Examples

This example shows a simple interactive session with the Terminal Interface.

display pod_command `<RETURN>`

pod_command keyboard `<RETURN>`

`cf <RETURN>`

`tsq <RETURN>`

`tcq <RETURN>`

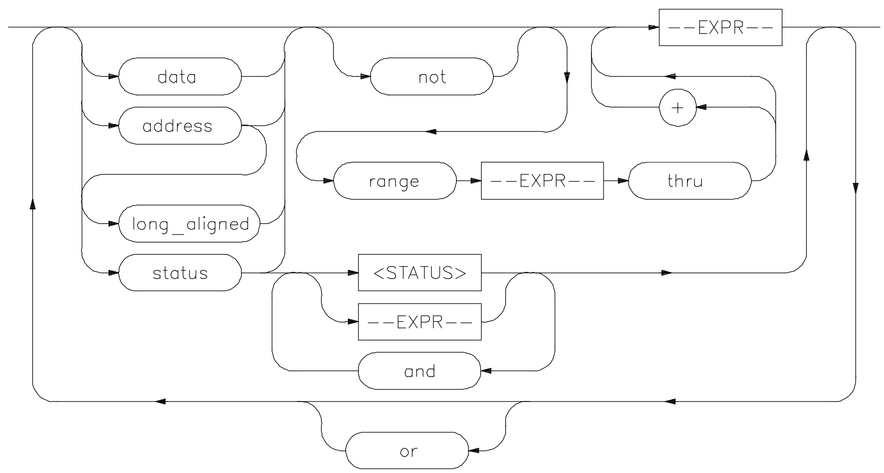
Enter **suspend** to return to the Graphical User Interface or Softkey Interface.

See Also

The **display pod_command** command.

The *68340 Emulator User's Guide for the Terminal Interface* and the Terminal Interface on-line help information.

QUALIFIER



The **QUALIFIER** parameter is used with **trace only**, **trace prestore**, **TRIGGER**, and **trace counting** to specify states captured during the trace measurement.

You may specify a range of states or specific states to be captured. You can continue to "or" states until the analyzer resources are depleted. You can use only one "range" statement in the entire **trace** command.

You can include "don't care numbers." These contain an "x" preceded and/or followed by a number. Some examples include 1fxxh, 17x7o, and 011xxx10b. "Don't care numbers" may be entered in binary, octal, or hexadecimal base.

Expression types are "address" when none is chosen. The default is to qualify on all states.

The parameters are as follows:

address — The value following this softkey is searched for on the lines that monitor the emulation processor's address bus. This is the default, and is therefore not required on the command line when specifying an address expression.

and — This lets you specify a combination of status and expression values when **status** is specified in the state specification.

data — The value following this softkey is searched for on the lines that monitor the emulation processor's data bus.

--EXPR-- — An expression is a combination of numeric values, symbols, operators, and parentheses, specifying an address, data, status value. See the EXPR syntax diagram for details.

long_aligned — Causes a mask to be applied to the address to force it to a long word boundary (least significant hex digit is 0, 4, 8, or C). This is necessary because the emulation-bus analyzer may not otherwise see the address on the address bus due to the way the processor fetches instructions.

not — This specifies that the analyzer will search for the logical "not" of a specified state (this includes any address that is not in the specified state).

or — This option allows you to specify multiple states to be captured during a trace measurement.

range — This indicates a range of addresses to be specified (--EXPR-- thru --EXPR--).

status — The value following this softkey is searched for on the lines that monitor emulation processor signals other than the address and data lines.

<STATUS> — This prompts you to enter a status value in the command line. Status values can be entered from softkeys or typed into the keyboard. Numeric values may be entered using symbols, operators, and parentheses to specify a status value. See the EXPR syntax diagram.

Refer to the "Qualifying Trigger and Store Conditions" section in the "Using the Emulation Analyzer" chapter for a list of the predefined values that can be assigned as status state qualifiers.

thru — This indicates that the following address expression is the upper address in a range.

Examples

trace only address `mod_name:read_input <RETURN>`

trace only address range `mod_name:read_input` ***thru*** `output <RETURN>`

trace only address range `mod_name:clear` ***thru*** `read_input <RETURN>`

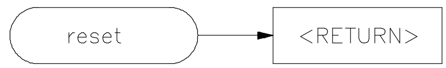
trace before status write <RETURN>

trace about address 1000H ***status write*** <RETURN>

See Also

The **trace** command.

reset



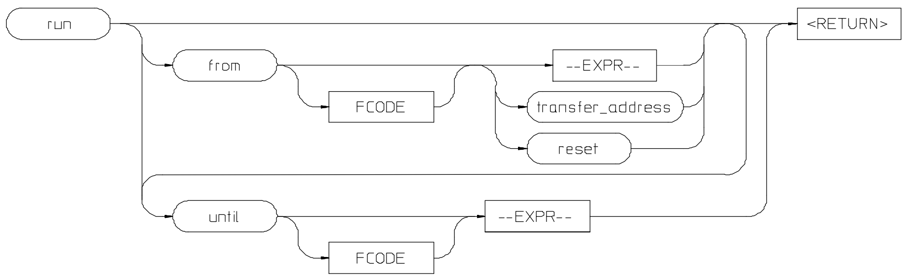
This command suspends target system operation and reestablishes initial emulator operating parameters, such as reloading control registers.

The reset signal is latched when the reset command is executed and released by either the **run** or **break** command.

See Also

The **break** and **run** commands.

run



This command causes the emulator to execute a program.

If the processor is in a reset state, **run** will cause the reset to be released.

If the **from** parameter and an address is specified, the processor will start running your program at that address. Otherwise, the run will occur from the address currently stored in the processor's program counter.

A **run from reset** command will reset the processor and then allow it to run. It is equivalent to entering a **reset** command followed by a **run** command.

If the emulator is configured to participate in the READY signal on the CMB, then this emulator will release the READY signal so that it will go TRUE if all other HP 64700 emulators participating on that signal are also ready. See the **cmb_execute** command description.

Qualifying a run command with an **until** parameter causes a software breakpoint to be set before the program is run.

If you omit the address option (**--**EXPR**--**), the emulator begins program execution at the current address specified by the emulation processor program counter. If an absolute file containing a transfer address has just been loaded, execution starts at that address.

The parameters are as follows:

--EXPR-- An expression is a combination of numeric values, symbols, operators, and parentheses, specifying a memory address. See the EXPR syntax diagram.

FCODE	The function code used to define the address space being referenced. See the syntax diagram for FCODE to see a list of the function codes available and for an explanation of those codes.
from	This specifies the address from which program execution is to begin.
reset	This option resets the processor prior to running from the reset vector address.
transfer_address	This represents the starting address of the program loaded into emulation or target memory. The transfer address is defined in the linker map and is part of the symbol database associated with the absolute file.
until	Causes a software breakpoint to be set at the specified address before the program is run.

Examples

run `<RETURN>`

run from `810H <RETURN>`

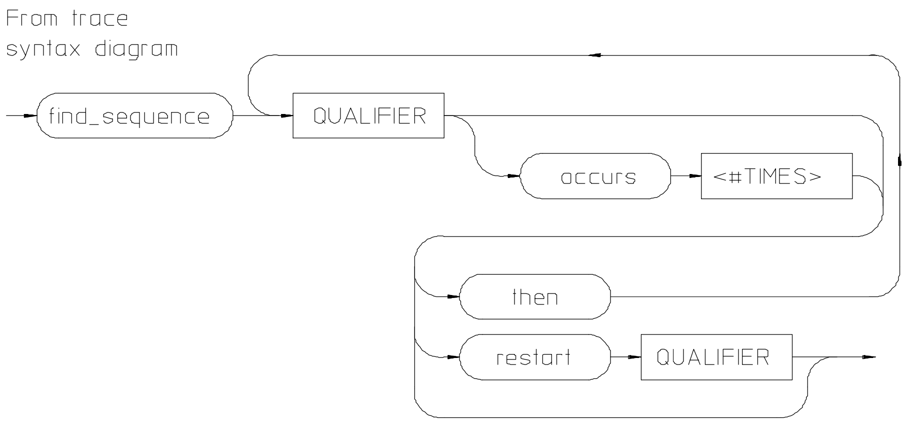
run from `COLD_START <RETURN>`

run from transfer_address until `910H <RETURN>`

See Also

The **step** command.

SEQUENCING



Lets you specify complex branching activity that must be satisfied to trigger the analyzer.

Sequencing provides you with parameters for the **trace** command that let you define branching conditions for the analyzer trigger.

You are limited to a total of seven sequence terms, including the trigger, if no windowing specification is given. If windowing is selected, you are limited to a total of four sequence terms.

The analyzer default is no sequencing terms. If you select the sequencer using the find_sequence parameter, you must specify at least one qualifying sequence term.

The parameters are as follows:

find_sequence — Specifies that you want to use the analysis sequencer. You must enter at least one qualifier.

QUALIFIER — Specifies the address, data, status value or value range that will satisfy this sequence term if looking for a sequence (find_sequence), or will restart at the beginning of the sequence (restart). See the QUALIFIER syntax pages for further information.

occurs — Selects the number of times a particular qualifier must be found before the analyzer proceeds to the next sequence term or the trigger term. This option is not available when trace windowing is in use. See the **WINDOW** syntax pages.

<#TIMES> Prompts you for the number of times a qualifier must be found.

then Allows you to add multiple sequence terms, each with its own qualifier and occurrence count.

restart Selects global restart. If the analyzer finds the restart qualifier while searching for a sequence term, the sequencer is reset and searching begins for the first sequence term.

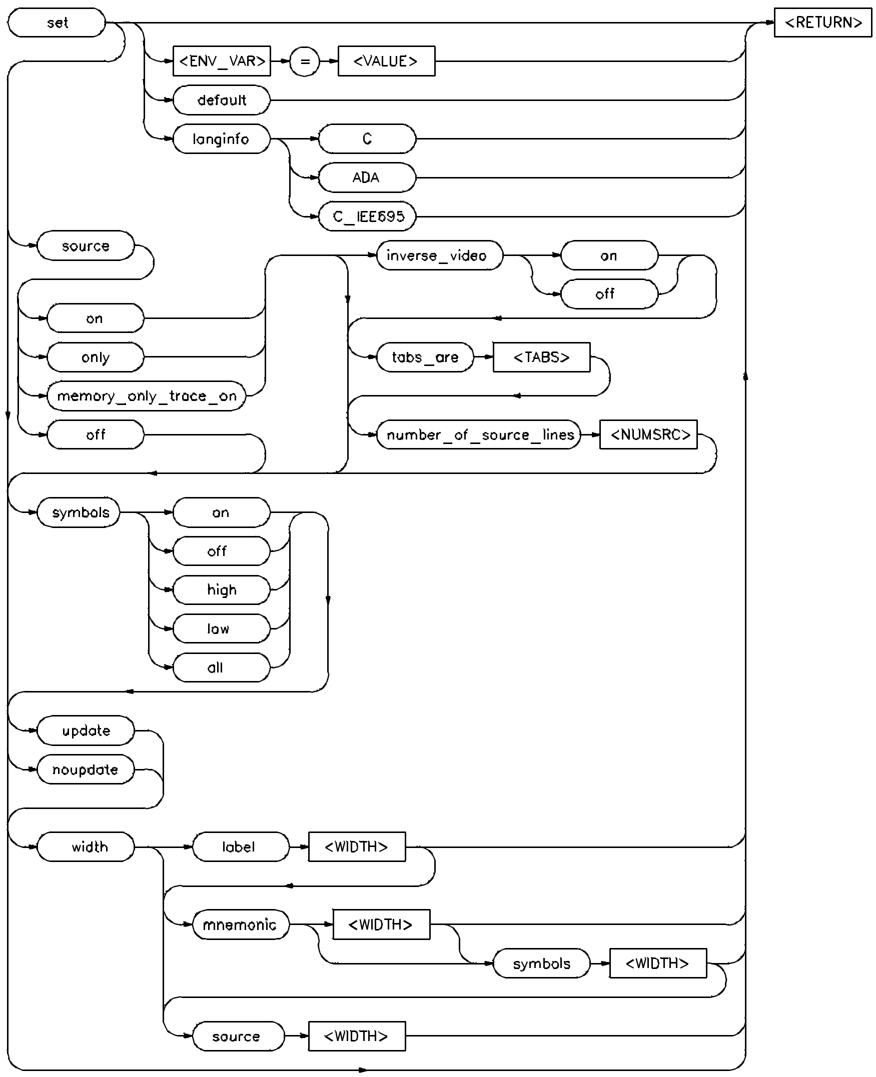
Examples

```
trace find_sequence Caller_3 then Write_Num restart
anly."anly.c": line 57 trigger after Results+0c4h
<RETURN>
```

See Also

The **trace** command and the QUALIFIER and WINDOW syntax descriptions.

set



Controls the display format for the data, memory, register, software breakpoint, and trace displays. With the **set** command, you can adjust the display format results for

various measurements, making them easier to read and interpret. Formatting of source lines, symbol display selection and width, and update after measurement can be defined to your needs.

The **display** command uses the **set** command specifications to format measurement results for the display window.

Another option to the set command, **<ENV_VAR> = <VALUE>**, allows you to set and export system variables to the UNIX environment.

The default display format parameters are the same as those set by the commands:

set update

set source off symbols off

You can return the display format to this state by entering:

set default

The parameters are as follows:

default — This option restores all the set options to their default settings.

<ENV_VAR> — Specifies the name of a UNIX environment variable to be set.

= — The equals sign is used to equate the <ENV_VAR> parameter to a particular value represented by <VALUE>.

inverse video

- off — This displays source lines in normal video.
- on — This highlights the source lines on the screen (dark characters on light background) to differentiate the source lines from other data on the screen.

langinfo — In certain languages, you may have symbols with the same names but different types. For example, in IEEE695, you may have a file named main.c and a procedure named main. SRU would identify these as main(module) and main(procedure). The command **display local_symbols_in main** would cause an error message to appear (Ambiguous symbol: main(procedure, module)). Users of C tend to think the procedure is important and users of ADA tend to think the module is important. By entering "langinfo" and "C", SRU will interpret the above command to be **main(procedure)**. With langinfo ADA, SRU will interpret the above command to be **main(module)**.

C — Identifies ANSI C as the language so SRU can use the C hierarchy to disambiguate symbols.

ADA — Identifies ADA as the language so SRU can use the ADA hierarchy to disambiguate symbols.

C_IEE695 — Identifies C_IEEE-695 as the language so SRU can use the C_IEEE-695 hierarchy to disambiguate symbols.

Note An alternate method for making the langinfo specification is to use the environment variable, HP64SYMORDER. By making the following entry in your **.profile**, the langinfo setting will always be C, for example.

```
$ HP64SYMORDER=C   # I want to use the C disambiguating
                   # hierarchy
$ export HP64SYMORDER   # let children processes know
                        # about it
```

noupdate — When using multiple windows or terminals, and specifying this option, the display buffer in that window or terminal will not update when a new measurement completes. Displays showing memory contents are not updated when a command executes that could have caused the values in memory to change (modify memory, load, etc.).

number_of_ source_lines — This allows you to specify the number of source lines displayed for the actual processor instructions to which they correlate. Only source lines up to the previous actual source line will be displayed. Using this option, you can specify how many comment lines are displayed preceding the actual source line. The default value is 5.

<NUMSRC> — This prompts you for the number of source lines to be displayed. Values in the range 1 through 50 may be entered.

source

memory_only_ trace_on — This provides a way to default the memory and trace displays to a setting that HP believes is the nicest possible formats for memory and trace displays. Parameters such as "source on/only", number of source lines to show, display width, and turning symbols on are all governed by this one selection. With this selection, memory displays will show the maximum available source lines preceding each block of code, and trace lists will show five source lines preceding trace data.

off — This option prevents inclusion of source lines in the trace and memory mnemonic display lists.

on — This option displays source program lines preceding actual processor instructions with which they correlate. This enables you to correlate processor instructions with your source program code. The option works for both the trace list and memory mnemonic displays.

only — This option displays only source lines. Processor instructions are only displayed in memory mnemonic if no source lines correspond to the instructions. Processor instructions are never displayed in the trace list.

symbols

off — This prevents symbol display.

on — This displays symbols. This option works for the trace list, memory, software breakpoints, and register step mnemonics.

high — Displays only high level symbols, such as those available from a compiler. See the *Symbolic Retrieval Utilities User's Guide* for a detailed discussion of symbols.

low — Displays only low level symbols, such as those generated internally by a compiler, or an assembly symbol.

all — Displays all symbols.

tabs_are — This option allows you to define the number of spaces inserted for tab characters in the source listing.

<TABS> — Prompts you for the number of spaces to use in replacing the tab character. Values in the range of 2 through 15 may be entered.

update — When using multiple windows or terminals, and specifying this option, the display buffer in that window or terminal will be updated when a new measurement completes. This is the default. Note that for displays that show memory contents, the values will be updated when a command executes that changes memory contents (such as modify memory, load, and so on).

<VALUE> — Specifies the logical value to which a particular UNIX environment variable is to be set.

width

source — This allows you to specify the width (in columns) of the source lines in the memory mnemonic display. To adjust the width of the source lines in the trace display, increase the widths of the label and/or mnemonic fields.

label — This lets you specify the address width (in columns) of the address field in the trace list or label (symbols) field in any of the other displays.

mnemonic — This lets you specify the width (in columns) of the mnemonic field in memory mnemonics, trace list and register step mnemonics displays. It also changes the width of the status field in the trace list.

symbols — This lets you specify the maximum width of symbols in the mnemonic field of the trace list, memory mnemonic, and register step mnemonic displays.

<WIDTH> — This prompts you for the column width of the source, label, mnemonic, or symbols field.

Note that <CTRL>f and <CTRL>g may be used to shift the display left or right to display information which is off the screen.

Examples

set source on inverse_video on tabs_are `2 <RETURN>`

set symbols on width label `30` ***mnemonic*** `20 <RETURN>`

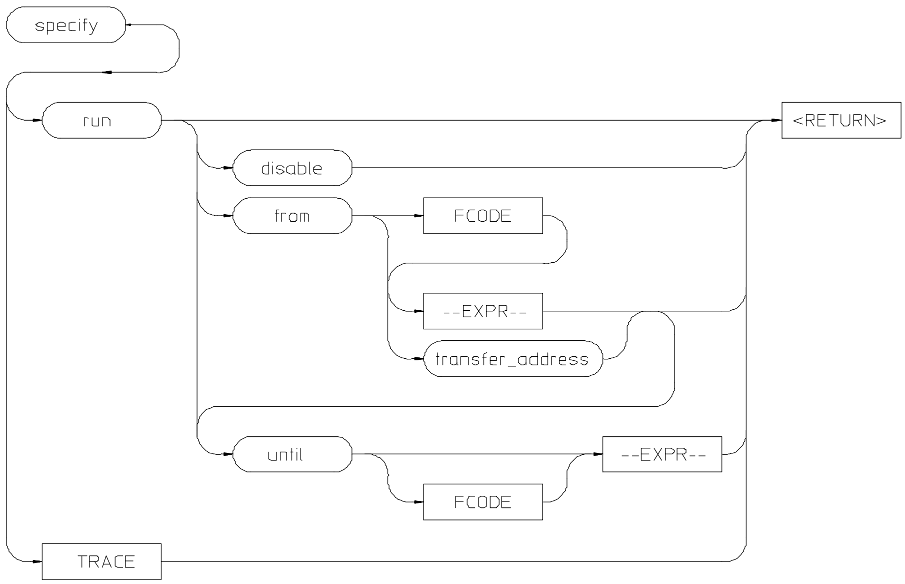
set `PRINTER = "lp -s" <RETURN>`

set `HP64KSYMBPATH=".file1:proc1`
`.file2:proc2:code_block_1" <RETURN>`

See Also

The **display data**, **display memory**, **display software_breakpoints**, and **display trace** commands.

specify



This command prepares a **run** or **trace** command for execution, and is used with the **cmb_execute** command.

When you precede a **run** or **trace** command with **specify**, the system does not execute your command immediately. Instead, it waits until until an EXECUTE signal is received from the Coordinated Measurement Bus or until you enter a **cmb_execute** command.

If the processor is reset and no address is specified, a **cmb_execute** command will run the processor from the "reset" condition.

Note that the **run** specification is active until you enter **specify run disable**. The trace specification is active until you enter another **trace** command without the **specify** prefix.

The emulator will run from the current program counter address if no address is specified in the command.

The parameters are as follows:

disable — This option turns off the specify condition of the **run** process.

from

--EXPR-- — This is used with the **specify run from** command. An expression is a combination of numeric values, symbols, operators, and parentheses, specifying a memory address. See the EXPR syntax diagram.

FCODE — The function code used to define the address space being referenced. See the syntax diagram for FCODE to see a list of the function codes available and for an explanation of those codes.

transfer_address — This is used with the **specify run from** command, and represents the address from which the program will begin running.

run — This option specifies that the emulator will run from either an expression or from the transfer address when a CMB EXECUTE signal is received.

TRACE — This option specifies that a trace measurement will be taken when a CMB EXECUTE signal is received.

until — Specifies an address where program execution is to stop. The emulator will set a software breakpoint at this address and stop execution of your program when it reaches this address and enter the monitor.

Examples

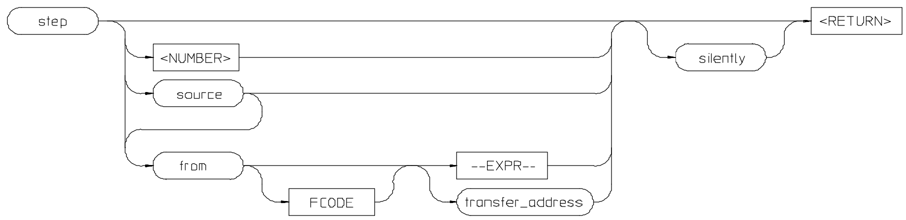
specify run from START <RETURN>

specify trace after address 1234H <RETURN>

See Also

The **cmb_execute** command.

step



The **step** command allows sequential analysis of program instructions by causing the emulation processor to execute a specified number of assembly instructions or source lines.

You can display the contents of the processor registers, trace memory, and emulation or target memory after each **step** command.

Source line stepping is implemented by single stepping assembly instructions until the next PC is beyond the address range of the current source line. When attempting source line stepping on assembly code (with no associated source line), stepping will complete when a source line is found. Therefore, stepping only assembly code may step forever. To abort stepping, press <CTRL>c.

When displaying memory mnemonic and stepping, the next instruction that will step is highlighted. The memory mnemonic display autopages to the new address if the next PC goes outside of the currently displayed address range. This feature works even if stepping is performed in a different emulation window than one displaying memory mnemonic.

If no value is entered for <NUMBER> times, only one **step** instruction is executed each time you press <RETURN>. Multiple instructions can be executed by holding down the <RETURN> key. Also, the default step is for assembly code lines, not source code lines.

If the **from** address option (defined by **--EXPR--** or transfer_address) is omitted, stepping begins at the next program counter address.

The parameters are as follows:

--EXPR-- An expression is a combination of numeric values, symbols, operators, and parentheses specifying a memory address. See the EXPR syntax diagram.

FCODE The function code used to define the address space being referenced. See the syntax diagram for FCODE to see a list of the function codes available and for an explanation of those codes.

from Use this option to specify the address from which program stepping begins.

<NUMBER> This defines the number of instructions that will be executed by the **step** command. The number of instructions to be executed can be entered in binary (B), octal (O or Q), decimal (D), or hexadecimal (H) notation.

silently This option updates the register step mnemonic only after stepping is complete. This will speed up stepping of many instructions. The default is to update the register step mnemonic after each assembly instruction (or source line) executes (if stepping is performed in the same window as the register display).

transfer_address This represents the starting address of the program you loaded into emulation or target memory. The transfer_address is defined in the linker map.

source This option performs stepping on source lines.

Examples

step `<RETURN>`

step from `810H <RETURN>`

step `20` ***from*** `0A0H <RETURN>`

step `5` ***source*** `<RETURN>`

step `20` ***silently*** `<RETURN>`

step `4` ***from*** `main <RETURN>`

See Also

The **display registers**, **display memory mnemonic**, and **set symbols** commands.

stop_trace

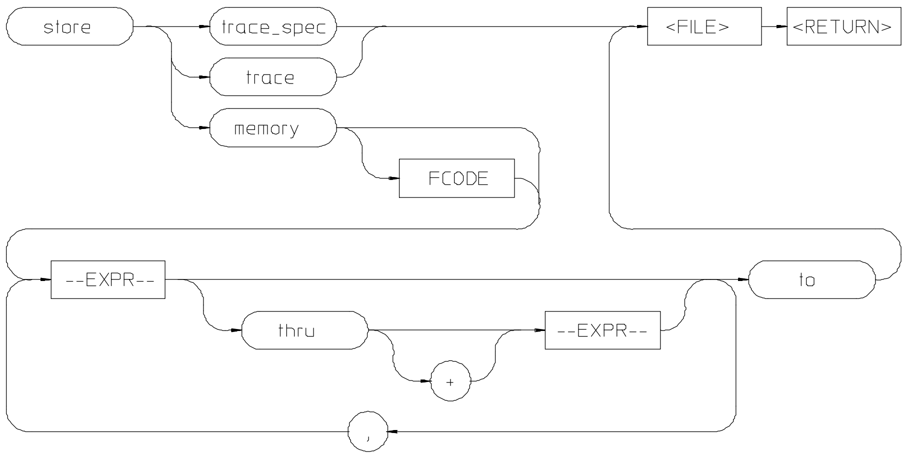
This command terminates the current trace and stops execution of the current measurement.

The analyzer stops searching for trigger and trace states. If trace memory is empty (no states acquired), nothing will be displayed.

See Also

The **trace** command.

store



This command lets you save the contents of specific memory locations in an absolute file. You also can save trace memory contents in a trace file.

The **store** command creates a new file with the name you specify, if there is not already an absolute file with the same name. If a file represented by <FILE> already exists, you must decide whether to keep or delete the old file. If you respond with **yes** to the prompt, the new file replaces the old one. If you respond with **no**, the **store** command is canceled and no data is stored.

The transfer address of the absolute file is set to zero.

The parameters are as follows:

--EXPR-- This is a combination of numeric values, symbols, operators, and parentheses, specifying a memory address. See the EXPR syntax diagram.

FCODE The function code used to define the address space being referenced. See the syntax diagram for FCODE to see a list of the function codes available and for an explanation of those codes.

<FILE> This represents a file name you specify for the absolute file identifier or trace file where data is to be stored. If you want to name a file beginning with a number, you must precede the file name with a backslash (\) so the system will recognize it as a file name.

memory — This causes selected memory locations to be stored in the specified HP64000 format file with a **.X** extension.

thru — This allows you to specify that ranges of memory be stored.

to — Use this in the **store memory** command to separate memory locations from the file identifier.

trace — This option causes the current trace data to be stored in the specified file with a **.TR** extension.

trace_spec — This option stores the current trace specification in the specified file with a **.TS** extension.

, — A comma separates memory expressions in the command line.

Examples

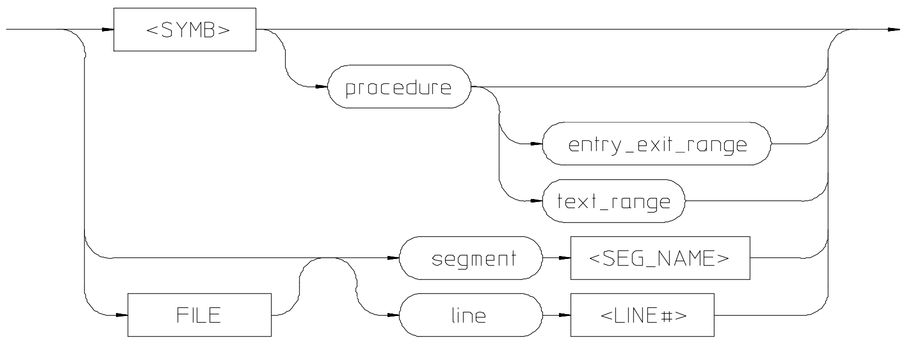
```
store memory 800H thru 20FFH to TEMP2 <RETURN>

store memory EXEC thru DONE to \12.10 <RETURN>

store trace TRACE <RETURN>

store trace_spec TRACE <RETURN>
```

See Also

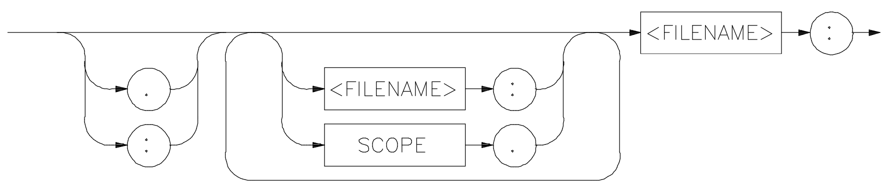
The **display memory**, **display trace**, and **load** commands.

--SYMB--

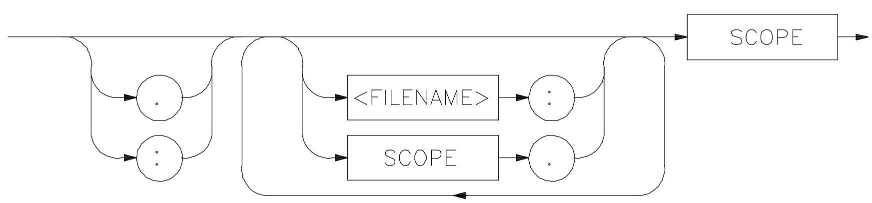
--SYMB--



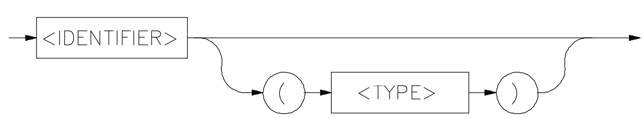
FILE



<SYMB>



SCOPE



This parameter is a symbolic reference to an address, address range, file, or other value. Symbols may be:

- Combinations of paths, filenames, and identifiers defining a scope, or referencing a particular identifier or location (including procedure entry and exit points).
- Combinations of paths, filenames, and line numbers referencing a particular source line.
- Combinations of paths, filenames, and segment identifiers identifying a particular PROG, DATA or COMN segment or a user-defined segment.

The Symbolic Retrieval Utilities (SRU) handle symbol scoping and referencing. These utilities build trees to identify unique symbol scopes.

If you use the SRU utilities to build a symbol database before entering the emulation environment, the measurements involving a particular symbol request will occur immediately. If you then change a module and reenter the emulation environment without rebuilding the symbol database, the emulation software rebuilds the changed portions of the database in increments as necessary.

Further information regarding the SRU and symbol handling is available in the *Symbolic Retrieval Utilities User's Guide.* Also refer to that manual for information on the **HP64KSYMBPATH** environment variable.

The last symbol specified in a **display local_symbols_in --SYMB--** command, or with the **cws** command, is the default symbol scope. The default is "none" if no current working symbol was set in the current emulation session.

You also can specify the current working symbol by typing the cws command on the command line and following it with a symbol name. The **pws** command displays the current working symbol on the status line.

Display memory mnemonic also can modify the current working symbol.

Note that if no default file was defined by executing the command **display local_symbols_in --SYMB--**, or with the **cws** command, a source file name (**<FILE>**) must be specified with each local symbol in a command line.

The parameters are as follows:

entry_exit_range — The range of addresses beginning with the entry point and ending with the return instruction. The entry point is the address used by other files when they call this procedure.

<FILENAME> — This is an UNIX path specifying a source file. If no file is specified, and the identifier referenced is not a global symbol in the executable file that was loaded, then the default file is assumed (the last absolute file specified by a display local_symbols_in command). A default file is only assumed when other parameters (such as **line**) in the **--SYMB--** specification expect a file.

line — This specifies that the following numeric value references a line number in the specified source file.

<LINE#> — Prompts you for the line number of the source file.

<IDENTIFIER> — Identifier is the name of an identifier as declared in the source file.

SCOPE — Scope is the name of the portion of the program where the specified identifier is defined or active (such as a procedure block).

segment — This indicates that the following string specifies a standard segment (such as PROG, DATA, or COMN) or a user-defined segment in the source file.

<SEG_NAME> — Prompts you for entry of the segment name.

text_range — The range of addresses beginning with the lowest address occupied by any code in the procedure and ending with the highest address occupied by any code in the procedure. Normally, the text_range will be the same as the entry_exit_range; some compilers may rearrange code so that the return instruction (for example) is not at the highest address in the range occupied by code of the procedure.

(<TYPE>) — When two identifier names are identical and have the same scope, you can distinguish between them by entering the type (in parentheses). Do not type a space between the identifier name and the type specification. The type will be one of the following:

filename — Specifies that the identifier is a source file.

fsegment — This provides an alternate way to reference a file segment in a command (example: **myfile.c:PROG(fsegment)**). It is better to use the keyword **segment** (example: **myfile.c: segment PROG**). The "segment" keyword is preferred because it will do scanning for PROG, Prog, prog, and other expressions of the program segment in

the example line. The **fsegment** keyword will only scan for the one expression (PROG).

module — These refer to module symbols. For most C compilers, these names derive from the source file name. For Ada, they are packages. Other language systems may allow user-defined module names.

procedure — Any procedure or function symbol. For languages that allow a change of scope without explicit naming, SRU assigns an identifier and tags it with type procedure.

prospecial — Prospecial symbols are symbols that were created by the HP SRU (such as entry, exit, and return). They are derived symbols, not intended for the product user. Typical prospecial symbols would be entry1, entry2, and entry3 in a procedure that has three possible entry points.

static — Static symbols, which includes global variables. The logical address of these symbols will not change.

task — Task symbols, which are specifically defined by the processor and language system in use.

: — A colon is used to specify the UNIX file path from the line, segment, or symbol specifier. When following the file name with a line or segment selection, there must be a space after the colon. For a symbol, there must not be a space after the colon.

Examples

The following short C code example should help illustrate how symbols are maintained by SRU and referenced in your emulation commands.

File /users/dave/control.c:

```
int *port_one;
main ()
{
int port_value;

  port_ptr = port_one;
  port_value = 10;

  process_port (port_ptr, port_value);
} /* end main */
```

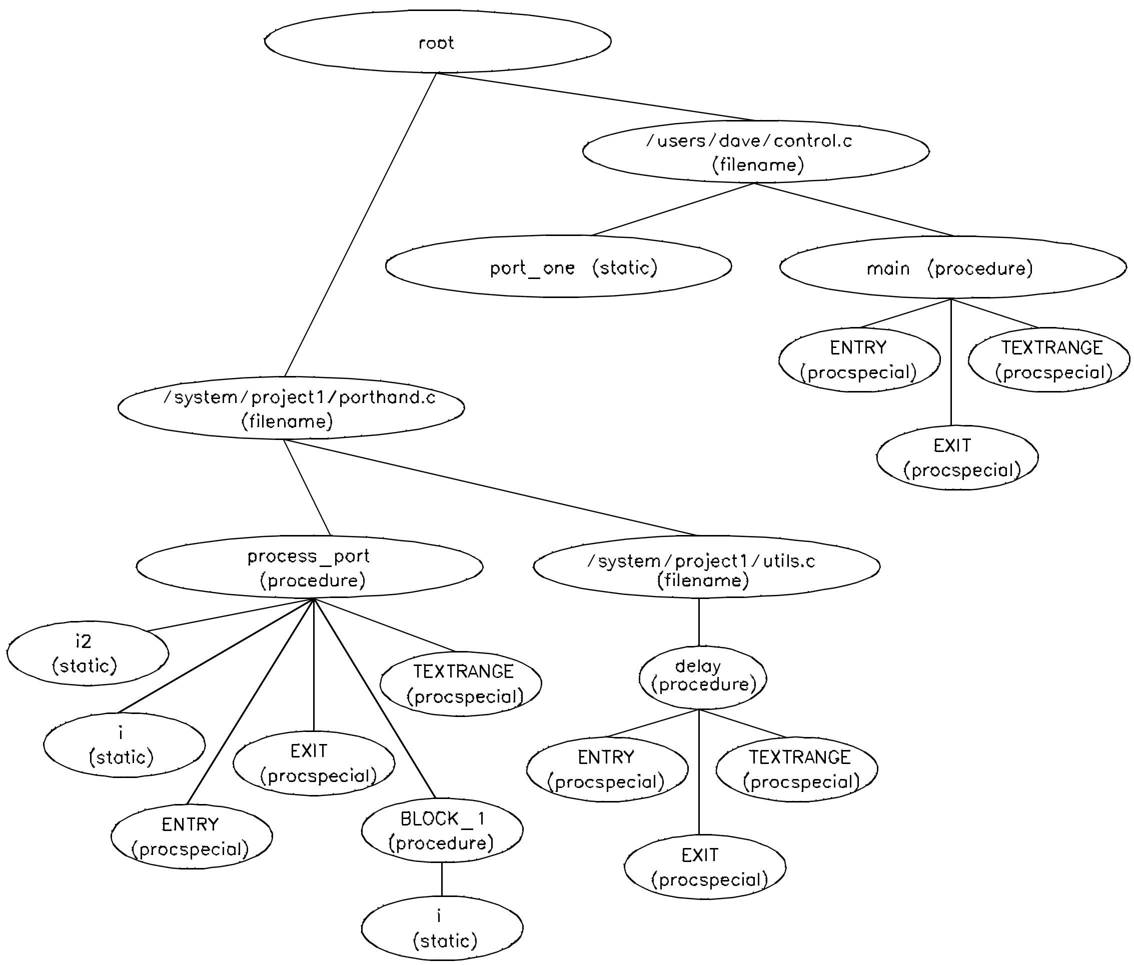
File /system/project1/porthand.c:

```
#include "utils.c"

void process_port (int *port_num, int port_data)
{
static int i;
static int i2;

  for (i = 0; i <= 64; i++) {
    i2 = i * 2;
    *port_num = port_data + i2;
    delay();
      {
      static int i;
      i = 3;
      port_data = port_data + i;
      }
    }
} /* end of process_port */
```

File /system/project1/utils.c:

```
delay()
{
int i,j;
int waste_time;

  for (i = 0; i <= 256000; i++)
    for (j = 0; j <= 256000; j++)
      waste_time = 0;
} /* end delay */
```

The symbol tree as built by SRU might appear as follows, depending on the object module format and compiler used:



Note that SRU does not build tree nodes for variables that are dynamically allocated on the stack at run-time, such as i and j within the delay () procedure.

SRU has no way of knowing where these variables will be at run time and therefore cannot build a corresponding symbol tree entry with run time address.

Here are some examples of referencing different symbols in the above programs:

```
control.c:main

control.c:port_one

porthand.c:utils.c:delay
```

The last example above only works with IEEE-695 object module format; the HP object module format does not support referencing of include files that generate program code.

```
porthand.c:process_port.i

porthand.c:process_port.BLOCK_1.i
```

Notice how you can reference different variables with matching identifiers by specifying the complete scope. You also can save typing by specifying a scope with cws. For example, if you are making many measurements involving symbols in the file porthand.c, you could specify:

```
cws porthand.c:process_port
```

Then:

```
i

BLOCK_1.i
```

are prefixed with porthand.c: process_port before the database lookup.

If a symbol search with the current working symbol prefix is unsuccessful, the last scope on the current working symbol is stripped. The symbol you specified is then retested with the modified current working symbol. Note that this does not change the actual current working symbol.

For example, if you set the current working symbol as

```
cws porthand.c:process_port.BLOCK_1
```

and made a reference to symbol i2, the retrieval utilities attempt to find a symbol called

```
porthand.c:process_port.BLOCK_1.i2
```

which would not be found. The symbol utilities would then strip BLOCK_1 from the current working symbol, yielding

```
porthand.c:process_port.i2
```

which is a valid symbol.

You also can specify the symbol type if conflicts arise. Although not shown in the tree, assume that a procedure called port_one is also defined in control.c. This would conflict with the identifier port_one which declares an integer pointer. SRU can resolve the difference. You must specify:

```
control.c:port_one(static)
```

to reference the variable, and

```
control.c:port_one(procedure)
```

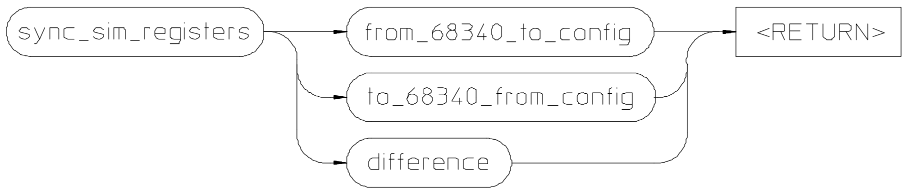
to reference the procedure address.

See Also

The **copy local_symbols_in** and **display local_symbols_in** commands.

Also refer to the *Symbolic Retrieval Utilities User's Guide* for further information on symbols.

sync_sim_registers



The **sync_sim_registers** command synchronizes the 68340's system integration module (SIM) registers to the emulator's CF_SIM registers.

The parameters are as follows:

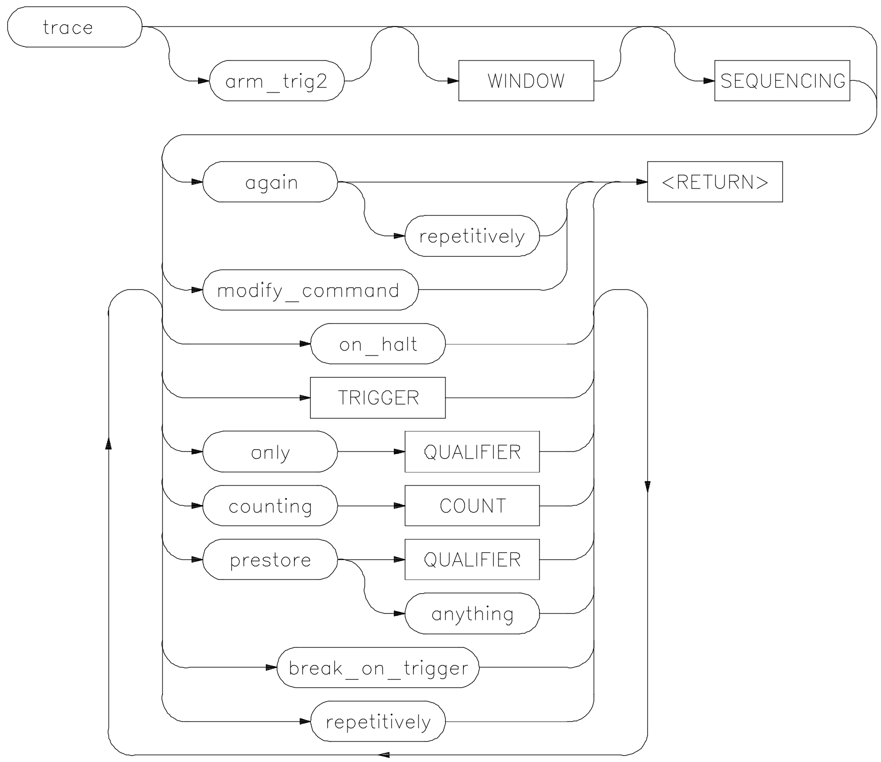
from_68340_to_config — Copies the microprocessor's SIM registers into the emulator's CF_SIM registers.

to_68340_from_config — Copies the emulator's CF_SIM registers into the microprocessor's SIM registers.

difference — Displays the differences between the microprocessor's SIM registers and the emulator's CF_SIM registers.

See Also — The **modify register** commands.

trace



This command allows you to trace program execution using the emulation analyzer.

The options shown can be executed once for each **trace** command. Refer to the COUNT, QUALIFIER, SEQUENCING, TRIGGER, and WINDOW diagrams for details on setting up a trace.

You can perform analysis tasks either by starting a program run and then specifying the trace parameters, or by specifying the trace parameters first and then initiating the program run. Once a **trace** begins, the analyzer monitors the system busses of the emulation processor to detect the states specified in the **trace** command. The analyzer will trace any state, counting time by default.

When the trace specification is satisfied and trace memory is filled, a message will appear on the status line indicating the trace is complete. You can then use display trace to display the contents of the trace memory. If a previous trace list is on screen, the current trace automatically updates the display. If the trace memory contents exceed the page size of the display, the <NEXT>, <PREV>, <Up arrow>, or <Down arrow> keys may be used to display all the trace memory contents. You also can press <CTRL>f and <CTRL>g to move the display left and right.

You can set up trigger and storage qualifications using the **specify trace** command. The analyzers will begin tracing when a **cmb_execute** command executes, which causes an EXECUTE signal on the Coordinated Measurement Bus.

The analyzer will trace any state by default.

The parameters are as follows:

again — This option repeats the previous trace measurement. It also begins a trace measurement with a newly loaded trace specification. (Using **trace** without the **again** parameter will start a trace with the default specification rather than the loaded specification.)

anything — This causes the analyzer to capture any type of information.

arm_trig2 — This option allows you to specify the external trigger as a trace qualifier, for coordinating measurements between multiple HP 64700s, or an HP 64700 and another instrument.

Before arm_trig2 can appear as an option, you must modify the emulation configuration interactive measurement specification. When doing this, you must specify that either BNC or CMBT drive trig2, and that the analyzer receive trig2. See the chapter on "Making Coordinated Measurements" for more information.

break_on_trigger — This stops target system program execution when the trigger is found. The emulator begins execution in the emulation monitor. When using this option, the **on_halt** option cannot be included in the command.

COUNT — This specifies whether time or state occurrences, or nothing, will be counted during the trace. See the COUNT syntax diagram for details.

counting — This option specifies whether the analyzer will count time or occurrences of states during a trace, or whether the option is to be turned off.

modify_command — This recalls the last trace command that was executed.

on_halt

When using this option, the analyzer will continue to capture states until the emulation processor halts or until a **stop_trace** command is executed. When this option is used, the **break_on_trigger**, **repetitively**, and **TRIGGER** options cannot be included in the command.

only

This option allows you to qualify the states that are stored, as defined by **QUALIFIER**.

prestore

This option instructs the analyzer to save specific states that occur prior to states that are stored (as specified with the "only" option).

QUALIFIER

This determines which of the traced states will be stored or prestored in the trace memory for display upon completion of the trace. Events can be selectively saved by using **trace only** to enter the specific events to be saved. When this is used, only the indicated states are stored in the trace memory. See the QUALIFIER syntax.

repetitively

This initiates a new trace after the results of the previous trace are displayed. The trace will continue until a **stop_trace** or a new **trace** command is issued. When using this option, you cannot use the **on_halt** option.

SEQUENCING

Allows you to specify up to seven sequence terms including the trigger. The analyzer must find each of these terms in the given order before searching for the trigger. You are limited to four sequence terms if windowing is enabled. See the **SEQUENCING** syntax pages for more details.

TRIGGER

This represents the event on the emulation bus to be used as the starting, ending, or centering event for the trace. See the **TRIGGER** syntax diagram. When using this option, you cannot include the **on_halt** option.

WINDOW

Selectively enables and disables analyzer operation based upon independent enable and disable terms. This can be used as a simple storage qualifier. Or, you may use it to further qualify complex trigger specifications. See the **WINDOW** syntax pages for details.

Examples

trace after `1000H <RETURN>`

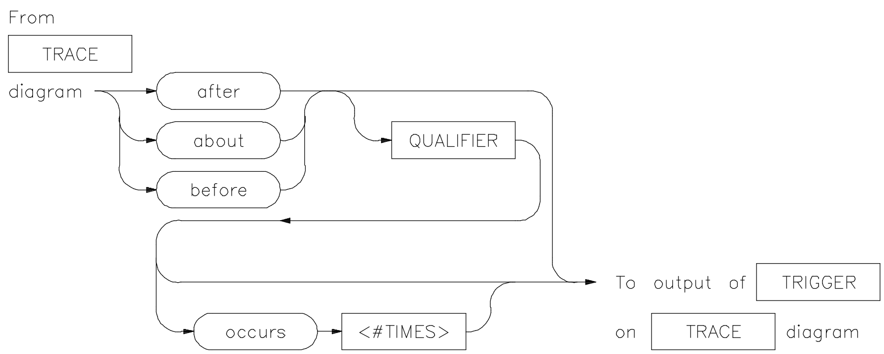
trace only address range `1000H` ***thru*** `1004H <RETURN>`

trace after address `1000H` ***occurs*** `2` ***only address range*** `1000H` ***thru*** `1004H` ***break_on_trigger*** `<RETURN>`

See Also

The **copy trace**, **display trace**, **load trace**, **load trace_spec**, **specify trace**, **store trace**, and **store trace_spec** commands.

TRIGGER



This parameter lets you define where the analyzer will begin tracing program information during a trace measurement.

A trigger is a QUALIFIER. When you include the **occurs** option, you can specify the trigger to be a specific number of occurrences of a QUALIFIER (see the QUALIFIER syntax diagram).

The default is to trace after any state occurs once.

The parameters are as follows:

about — This option captures trace data leading to and following the trigger qualifier. The trigger is centered in the trace listing.

after — Trace data is acquired after the trigger qualifier is found.

before — Trace data is acquired prior to the trigger qualifier.

occurs — This specifies a number of qualifier occurrences of a range or state on which the analyzer is to trigger.

QUALIFIER — This determines which of the traced states will be stored in trace memory.

<#TIMES> — This prompts you to enter a number of qualifier occurrences.

Examples

trace after `MAIN <RETURN>`

trace after `1000H` ***then data*** `5 <RETURN>`

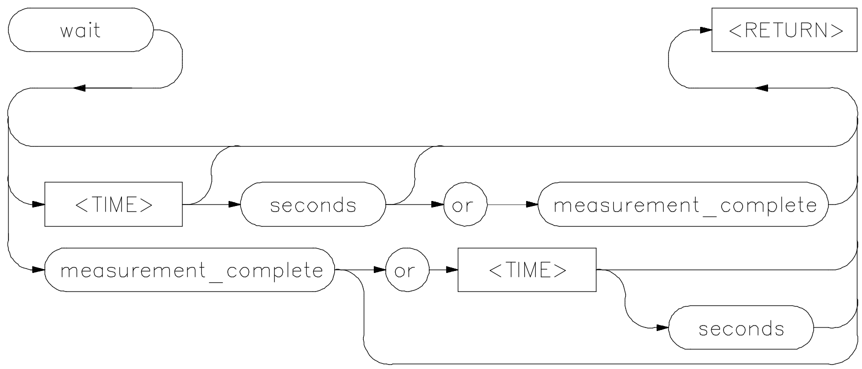
Also see the **trace** command examples.

See Also

The **trace** command.

Also, refer to the "Making Coordinated Measurements" chapter.

wait



This command allows you to present delays to the system.

The **wait** command can be an enhancement to a command file, or to normal operation at the main emulation level. Delays allow the emulation system and target processor time to reach a certain condition or state before executing the next emulation command.

The **wait** command does not appear on the softkey labels. You must type the **wait** command into the keyboard. After you type **wait**, the command parameters will be accessible through the softkeys.

If you issue a **wait** command without any other options, the system will pause until it receives a <CTRL>c signal.

Note that if **set intr <CTRL>c** was not executed on your system, <CTRL>c normally defaults to the backspace key. See your UNIX system administrator for more details regarding keyboard definitions.

Note that a **wait** command in a command file will cause execution of the command file to pause until a <CTRL>c signal is received, if <CTRL>c is defined as the interrupt signal. Subsequent commands in the command file will not execute while the command file is paused. You can verify whether the interrupt signal is defined as <CTRL>c by typing **set** at the system prompt.

The parameters are as follows:

measurement_complete — This causes the system to pause until a pending measurement completes (a trace data upload process completes), or until a <CTRL>c signal is received. If a measurement is not in progress, the **wait** command will complete immediately.

or — This causes the system to wait for a <CTRL>c signal or for a pending measurement to complete. Whichever occurs first will satisfy the condition.

seconds — This causes the system to pause for a specific number of seconds.

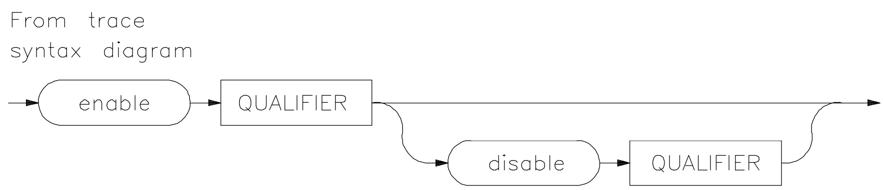
<TIME> — This prompts you for the number of seconds to insert for the delay.

Examples

wait <RETURN>

wait 5; ***wait measurement_complete*** <RETURN>

WINDOW



Lets you select which states are stored by the analyzer.

WINDOW allows you to selectively toggle analyzer operation. When enabled, the analyzer will recognize sequence terms, trigger terms, and will store states. When disabled, the analyzer is effectively off, and only looks for a particular enable term.

You specify windowing by selecting an enable qualifier term; the analyzer will trigger or store all states after this term is satisfied. If the disable term occurs after the analyzer is enabled, the analyzer will then stop storing states, and will not recognize trigger or sequence terms. You may specify only one enable term and one disable term.

The analyzer defaults to recognizing all states. If you specify enable, you must supply a qualifier term. If you then specify disable, you must specify a qualifier term.

The parameters are as follows:

disable Allows you to specify the term which will stop the analyzer from recognizing states once the enable term has been found.

enable Allows you to specify the term which will enable the analyzer to begin monitoring states.

QUALIFIER Specifies the actual address, data, status value or range of values that cause the analyzer to enable or disable recognition of states. Note that the enable qualifier can be different from the disable qualifier. Refer to the QUALIFIER syntax pages for further details on analyzer qualifier specification.

Examples

trace enable `_rand` ***disable*** `0ecch <RETURN>`

See Also

The **trace** command and the SEQUENCING and QUALIFIER syntax descriptions.

12

Error Messages

Error Messages

This chapter contains a list of error messages that may occur while operating the emulator and analyzer.

The *error log* records error messages received during the emulation session. You may want to display the error log to view the error messages. Sometimes several messages will be displayed for a single error to help you locate a problem quickly. To prevent overrun, the error log purges the oldest messages to make room for the new ones.

To display the error log:

`display error_log` `<RETURN>`

Error messages are grouped into the following categories:

- Graphical/Softkey Interface Messages - Unnumbered
- Graphical/Softkey Interface Messages - Numbered
- Terminal Interface Messages

Note that Terminal Interface messages are passed along to the Graphical User Interface (or Softkey Interface) and appear, with numbers, in the error log display.

Graphical/Softkey Interface Messages - Unnumbered

Address range too small for request - request truncated

Cause: Too small of an address range is specified in a modify memory command.

Action: Specify a larger memory range.

Cannot create module file:

Cause: Insufficient disk space for the module file.

Action: Check disk space under /usr/hp64000.

Cannot start. Ending previous session, try again

Cause: The host system could not start a new emulation session, and is ending the previous session.

Action: After the previous session has ended, try starting a new emulation session. If that fails, try "emul700 -u <logical name>" to unlock the emulator and cycle power, if needed.

Cannot start. Pod initialization failed

Cause: The host system could not start a new emulation session because it could not initialize the emulator.

Action: Cycle power on the emulator; verify that there are no red lights on the front of the emulator. You may need to run the Terminal Interface "pv" command to verify that the emulator is functioning properly before starting a new session.

Configuration not valid, restoring previous configuration
Configuration not valid, restoring default configuration

Cause: The modifications you tried to make to the emulator configuration are not valid, so the host system restored the previous configuration.

Action: See the "Configuring the Emulator" chapter for more information about the emulator configuration items and their settings.

Configuration process QUIT

Cause: The configuration process ended because <CTRL> "\" (SIGQUIT signal) was encountered. This is an easy way to exit configuration without saving any changes.

Action: Try starting the emulation session again. If the problem persists, you may need to cycle power on the emulator.

Connecting to <LOGICAL NAME>

Cause: This is a status message. The host system is making a communication connection to the emulator whose logical name is defined in /usr/hp64000/etc/64700tab.net or /usr/hp64000/etc/64700tab.

Continue load failed

Cause: The host system could not continue the previous emulation session because it could not load the continue file.

Action: Try again. If the failure continues, call your HP Service Representative.

Continuing previous session, continue file loaded

Cause: This is a status message. An emulation session which was ended earlier with the **end** command has been restarted. The host system reported that the session was continued (using settings from the previous session) and that the continue file loaded properly.

Continuing previous session, user interface defaulted

Cause: The previous emulation session was continued and the Softkey Interface was set to the default state.

Could not create default configuration

Cause: The host system could not create a default configuration for the emulation session.

Action: Check disk space under /usr/hp64000 and verify proper software installation.

Could not create <CONFIGURATION BINARY FILENAME>

Cause: The system could not create a binary emulation configuration file (file.EB).

Action: Check the file.EB write permission and verify that the specified directory exists and is writeable.

Could not exec configuration process

Cause: The host system could not fork the configuration process or could not execute the configuration process.

Action: Make sure that the host system is operating properly, and that all Softkey Interface files were loaded properly during the installation process. Try starting the emulation session again.

Could not load default configuration

Cause: The host system could not load the default configuration into the emulator.

Action: Cycle power on the emulator and run the Terminal Interface "pv" (performance verification) command on the emulator to verify that it is functioning properly. Also, verify proper software installation. If loading default configuration still fails, then call your HP 64000 representative.

<CONFIGURATION FILENAME> does not exist

Cause: The configuration file you are trying to load does not exist.

Action: Try the **load configuration** command again using a valid configuration file name.

Don’t care number unexpected

Cause: While defining an expression in your command, you included a don’t care number (a binary, octal, decimal, or hexadecimal number containing "x"), which was not expected. Don’t care numbers are not valid for all commands. See the EXPR command syntax for more information about expressions.

Emulation analyzer defaulted to delete label

Cause: Analyzer trace labels were changed or modified while labels were in use in the trace specification.

Action: Enter the previous trace specification and try again.

Emul700dmn continuation failed

Cause: Communication between the emulator and the host system to continue the emulation session failed.

Action: Check the data communication switch settings on the rear panel of the HP 64700 series emulator. If necessary, refer to the *HP 64700 Installation/Service Guide*.

Emul700dmn executable not found

Cause: The emulation session could not begin because the host system could not locate the HP 64700 emulator daemon process executable.

Action: Make sure that software installation is correct. Then try starting the emulator again.

Emul700dmn failed to start

Cause: The emulation session could not begin because the host system could not start the HP 64700 emulator daemon process.

Action: Make sure there is sufficient disk space under /usr/hp64000. Make sure the host system is operating properly, that all Softkey Interface software has been loaded correctly, and the data communication switch settings on the emulator rear panel match the settings in the /usr/hp64000/etc/64700tab.net (or 64700tab) file.

Emul700dmn message too large
Emul700dmn message too small
Emul700dmn queue and/or semaphores missing
Emul700dmn queue failure
Emul700dmn error in file operation
Emul700dmn queue full

Cause: The HP 64700 emulator daemon process command was too large for the host system to process.

Action: You must press **end_release_system** to exit this emulation session completely; then start a new session. Make sure the host system is operating properly, that all Softkey Interface software has been loaded correctly, and the data communication switch settings on the emulator rear panel match the settings in the

/usr/hp64000/etc/64700tab.net (or 64700tab) file. You may have to cycle power and use **emul700 -u ,logical name>** to unlock the system.

Emul700dmn sem op failed, perhaps kernel limits too low

Cause: The host system could not start the emulation session; there may be too many processes running on the host system.

Action: Make sure the host system is operating properly, and is not overloaded with currently executing processes. Stop or remove some processes on the system. Also, verify that the semaphore capabilities have been installed in the UNIX kernel. Then try starting the emulation session again.

Emul700dmn version incompatible with this product

Cause: The emulation session could not begin because the version of the HP 64700 emulator daemon executable on host system is not compatible with the version of the Softkey Interface you are using.

Action: Make sure the software has been properly installed. Then try starting the emulator again.

<LOGICAL NAME>: End, continuing

Cause: This is a status message. The emulation session is being exited with the **end** command. When you restart the emulation session later, it will continue using the same settings as in the session you just ended. The emulator logical name is located in the /usr/hp64000/etc/64700tab.net (or 64700tab) file.

<LOGICAL NAME>: End, released

Cause: This is a status message. The emulation session is being exited with the **end release_system** command. When the session has ended, the emulator is released, meaning that others can access and use it. When you restart the emulation session later, the new session will use all default settings. The emulator logical name is located in the /usr/hp64000/etc/64700tab.net (or 64700tab) file.

Ending released

Cause: This is a status message. The emulation session is being exited with the **end release_system**. The emulator will be released for others to access and use it.

Error: display size is <LINES> lines by <COLUMNS> columns. It must be at least 24 by 80.

Cause: You tried to specify an incorrect window size.

Action: Set the window size accordingly, then start the emulation session. The size of the window must be a minimum of 24 lines (rows) by 80 columns to operate an emulation session.

Error in configuration process
Error starting configuration process

Cause: Unexpected configuration error.

Action: Verify proper software installation and call your HP 64000 representative.

Fatal error from function <ADDRESS OF FUNCTION>

Cause: This is an unexpected fatal system error.

Action: Cycle power on the emulator and start again. If this is a persistent problem, call your HP 64000 representative.

File could not be opened

Cause: You tried to store or load trace data to a file with incorrect permission. Or the analyzer could not find the file you specified, or else there were already too many files open when you entered your command.

Action: Check the directory and file for correct read and write permission. Specify a file that is accessible to the analyzer. Close the other files that are presently open.

File perf.out does not exist

Cause: You tried to execute the "restore" command to continue a previous software performance measurement, and the SPMT software found that no "performance_measurement_end" command was previously executed to create a file from which "restore" could be performed.

Action: Execute a new SPMT measurement.

File perf.out not generated by measurement software

Cause: The file named perf.out exists in the current directory, but it was not created by the "performance_measurement_end" command.

Action: Rename the old "perf.out" file, or move it to another directory.

HP64700 I/O channel semaphore failure: <string>

Cause: Semaphore (ipc) facility not installed.

Action: Reconfigure the kernel to add ipc facility.

HP 64700 I/O error; communications timeout

Cause: This is a communication failure.

Action: Check power to the emulator and check that all cables are connected properly. If you are using LAN and heavy LAN traffic is present, try setting the environment variable to HP64700TIMEOUT="30" (or larger if needed). The value is the number of seconds before timeout occurs. Then try running again.

HP64700 I/O error; connection timed out

Cause: A user abort occurred while attempting to connect via LAN.

Action: Possibly connecting to an emulator many miles away, be patient.

HP 64700 I/O error; power down detected

Cause: The emulator power was cycled.

Action: Do not do this during a user interface session; this may force the user interface to end immediately.

HP64700 I/O channel busy; communications timed out

Cause: The communications channel is in use for an unusually long period of time by another command.

Action: try again later.

Illegal status combination

Cause: You tried to specify combinations of status qualifiers in expressions incorrectly when entering commands.

Action: Refer to the "Emulator/Analyzer Interface Commands" chapter for information about syntax of commands.

Illegal symbol name

Cause: You tried to specify incorrect symbol names when entering commands.

Action: Specify correct symbol names. To see global symbol names, use the **display global_symbols** command. To see local symbol names, use the **display local_symbols_in <SYMB>** command.

Initialization failed

Cause: The emulator could not be initialized.

Action: Make sure your data communication switch settings are correct, and that all Softkey Interface software has been loaded properly. Cycle power on the emulator, then try starting up the emulation session again.

Initialization load failed

Cause: The emulator could not be initialized.

Action: Make sure your data communication switch settings are correct, and that all Softkey Interface software has been loaded properly. Cycle power on the emulator, then try starting up the emulation session again.

Initializing emulator with default configuration

Cause: This is a status message. The host system started the emulation session and initialized the emulator using the default configuration. The emulator is probably operating correctly.

Initializing user interface with default config file

Cause: This is a status message. The host system started the emulation session and Softkey Interface using the default configuration file. The emulator is probably operating correctly.

Insufficient emulation memory, memory map may be incomplete

Cause: You can map only the amount of emulation memory available in your emulator. Trying to map additional unavailable memory may cause information to be missing from your memory map.

Action: Modify your configuration and update the memory map to correctly reflect the amount of emulation memory available.

Invalid answer in <CONFIGURATION FILENAME> ignored

Cause: You must provide acceptable responses to questions in the configuration file (file.EA). The emulator ignored the incorrect response. Incorrect responses may appear in configuration files when you have saved the configuration to a file, edited it later, and tried reloading it into the emulator. This may also occur if you have loaded a configuration file that you created while using another emulator, and the response differs from the response required for this emulator.

Action: Examine your configuration file to check for inappropriate responses to configuration file questions.

Inverse assembly file <INVERSE ASSEMBLER FILENAME> could not be loaded
Inverse assembly file <INVERSE ASSEMBLER FILENAME> not found, <filename>
Inverse assembly not available

Cause: The file does not exist.

Action: Reload your interface and/or real-time operating system software.

Inverse assembly not available

Cause: The inverse assembler for your emulator is missing.

Action: Verify proper software installation.

Joining session already in progress, continue file loaded

Cause: This is a status message. When operating the emulator in multiple windows, a new emulation session is "joined" to a current session. In this case, the new session was able to continue because the continue file loaded properly.

Joining session already in progress, user interface defaulted

Cause: When operating the emulator in multiple windows, a new emulation session is "joined" to a current session. In this case, the new session used the user interface default selections.

Load aborted

Cause: While loading a file into the emulator, an event occurred that caused the host system to stop the load process.

Action: Use the **display error_log** command to view any errors. If the problem persists, make sure the host system and emulator are operating properly, and that you are trying to load an acceptable file. See the "Emulator/Analyzer Interface Commands" chapter for information about the **load** command.

Load completed with errors

Cause: While loading a file into the emulator, one or more events occurred that caused errors during the load process.

Action: Use the **display error_log** command to view any errors. You may need to modify the configuration and map memory before you load the file again. If the problem persists, make sure the host system and emulator are operating properly, and that you are trying to load an acceptable file.

Measurement system not found

Cause: You tried to end the current emulation session and select another measurement system module which could not be located by the host system.

Action: Either try the **end select measurement_system** command again or end and release the emulation session.

Memory allocation failed, ending released

Cause: This is a fatal system error because the emulation session was unable to allocate memory.

Action: You may need to reconfigure your UNIX kernel to increase the per process maximum memory limit and available swap space. Reboot your UNIX system and try starting a new session again.

Memory block list unreadable

Memory range overflow

Cause: A modify memory command is attempted that would cross physical 0.

Action: Limit the modify memory command to not overflow physical 0 or break the command into two separate modify commands.

No address label defined

Cause: The address trace label was somehow removed in the terminal interface using the **tlb** command.

Action: End session and start again.

No more processes may be attached to this session

Cause: You can operate an emulator in four windows. Each time you start the emulator in another window, a new process is attached to the current session.

Action: Do not try to use more than four windows. Once you have started the emulator in four windows, you have reached the maximum number of processes allowed for that emulator.

Not an absolute file
No absolute file: <file>
No absolute file, No database: <file>

Cause: You tried to load a file into the emulator that is not an executable or absolute file, so the host system stopped the load process.

Action: Try your command again, and make sure you specify a valid absolute file name to be loaded.

No symbols loaded

Cause: You tried to step through lines in the source file before symbols are loaded.

Action: Load symbols and try again, or use step with the "source" option (i.e. step assembly language program).

No valid trace data

Cause: You tried to store trace data before a trace was completed.

Action: Wait until valid trace data is available before attempting to store a trace.

Not a valid trace file - load aborted

Cause: You tried to load a file.TR that was not created by the emulation session.

Action: Only load trace data files that were created by the emulator.

Not compatible trace file - load aborted

Cause: You tried to load a file.TR that was created by another type of emulator.

Action: Only load trace data files that were created by the same type of emulator.

Number of lines not in range: 1 <= valid lines <= 50

Cause: You tried to enter a number of lines that was outside the range from 1 to 50.

Action: Try entering the command again using a valid number of lines.

Number of spaces not in range: 2 <= valid spaces <= 15

Cause: You tried to enter a number of spaces outside the range from 2 to 15.

Action: Try entering the command again using a valid number of spaces.

opcode extends beyond specified address range

Cause: Memory disassembly is attempted on an address range that is too small.

Action: Display memory mnemonic using a large address range, or no address range at all.

Perfinit - Absolute file (database) must be loaded line <LINE NUMBER>

Cause: No symbolic data base has been opened (or exists) for the target file when you executed the "performance_measurement_initialize" command.

Action: Make sure a data base has been loaded for the target file.

Perfinit - error in input file line <LINE NUMBER> invalid symbol

You included a "label" file name with your "performance_measurement_initialize" command, and that file contains an invalid symbol.

Action: Edit the file and correct the invalid symbol.

Perfinit - error in input file line <NUMBER>

Cause: You included an input file name with your "performance_measurement_initialize" command, and that file contains a syntax error.

Action: Edit the file and correct the syntax error.

Perfinit <—-EXPR—- ERROR> line <LINE NUMBER>

Perfinit - File could not be opened

Cause: You specified a file as an option to "performance_measurement_initialize", and the file you specified could not be found or opened by SPMT software.

Action: Make sure you entered the correct file name.

Perfinit - No events in file

Cause: You specified a file along with your "performance_measurement"initialize" command that contained no events. Any measurement displayed from this file will have NULL results.

Action: Either edit the file to add events, or use the default setup to start a new measurement.

perf.out file could not be opened - created

Cause: The performance analyzer failed to open or create a file named "perf.out" in response to your "performance_measurement_end" command.

Action: Free up some file space or correct the write permissions in your current working directory.

Performance tool must be initialized

Cause: You tried to make a performance measurement when the Software Performance Measurement Tool (SPMT) was not initialized.

Action: The Software Performance Measurement Tool (SPMT) must be initialized before making performance measurements on your software. Use the **performance_measurement_initialize** command to initialize the SPMT.

Performance tool not initialized

Cause: The Software Performance Measurement Tool (SPMT) has not been initialized.

Action: To make accurate activity or duration measurements on current data, use the **performance_measurement_initialize** command to initialize the SPMT before running a performance measurement.

Question file missing or invalid

Cause: Some of the Softkey User Interface files are missing or are corrupted.

Action: Reinstall the host software and try starting the emulation session again.

Range crosses segment boundary

Cause: On a segment offset processor, an address range is specified that would cross different segments.

Action: Break the memory command into multiple commands so that the address ranges start and end in the same segment.

Read memory failed at <PHYSICAL ADDRESS> - store aborted

Cause: While storing memory from the emulator to a file, a read memory error occurred.

Action: Use the **display error_log** command to view any errors. You may need to modify the configuration and map memory before storing the file again.

Session aborted

Cause: This will only happen when running multiple emulation windows and a fatal system error occurs.

Action: Find the window that caused the error and see the error message that it displayed. All the additional windows will simply state "session aborted". Cycle power on the emulator and enter **emul700 -u <logical name>** to make sure the emulator is unlocked.

Session cannot be continued, ending released

Cause: The emulation session is ending automatically because it could not be continued from the previous session. When the session has ended the emulator will be released, meaning that others can access and use it.

Action: When you restart the emulation session later, the new session will use all default settings.

Slave clock requires at least one edge

Cause: The analyzer has an invalid clock specification.

Action: Modify your configuration and try your command again.

Starting address greater than ending address

Cause: You specified a starting address that is greater than the ending address.

Action: Specify a starting address that is less than or equal to the ending address.

Starting new session, continue file loaded

Cause: This is a status message. The emulator was started using a new emulation session, and the continue file loaded properly.

Starting new session, user interface defaulted

Cause: The emulator was started using a new emulation session, and the user interface was set to default selections.

Action: Call your HP Service Representative.

Status unknown, run "emul700 -l <LOGICAL NAME>"

Cause: The host system cannot determine the status of the emulator.

Action: To verify communication between the emulator and the host system, and display the emulator status, enter the **emul700 -l <logical name>** command. The

emulator logical name is located in the /usr/hp64000/etc/64700tab.net (or 64700tab) file.

Stepping aborted; number steps completed: <STEPS TAKEN>

Cause: Stepping aborted because <CTRL>c or software breakpoint was hit, guarded memory was accessed, or some other kind of error occurred.

Action: See the error log display for any abnormal errors. Correct those errors and then step again.

Stepping complete

Cause: Stepping was completed successfully.

Step count must be 1 through 999

Cause: You tried to use a step count greater than 999.

Action: Use a step count less than 1000.

Symbols not accessible, symbol database not loaded

Cause: You specified a trace list with values expressed using symbols defined in the source code modules, such as source on, and the database file has not been loaded into emulation. Example: display trace symbols on.

Timeout in emul700dmn communication

Cause: The host system could not start the emulation session because the HP 64700 emulator process ran out of time before the emulator could start.

Action: You must press **end_release_system** to exit this emulation session completely; then start a new session. Make sure the host system is operating properly, that all Softkey Interface software has been loaded correctly, and the data communication switch settings on the emulator rear panel match the settings in the /usr/hp64000/etc/64700tab.net (or 64700tab) file.

Trace file not found

Cause: You tried to load trace data file that does not exist.

Action: Find the correct name and path of the trace data file and try again.

Unexpected message from emul700dmn

Cause: The host system could not start the emulation session because of an unexpected message from the HP 64700 emulator process command.

Action: You must press **end_release_system** to exit this emulation session completely; then start a new session. Make sure the host system is operating properly, that all Softkey Interface software has been loaded correctly, and the data communication switch settings on the emulator rear panel match the settings in the /usr/hp64000/etc/64700tab.net (or 64700tab) file.

Unknown expression type

Cause: While entering your command, you included an unknown expression type.

Action: See the EXPR command syntax for more information about expressions. Then try entering your command again with a known expression type.

Unload trace data failed

Cause: An unexpected error occurred while waiting for a trace to be completed.

Action: End and release the session, and then try again.

Wait time failure, could not determine system time

Cause: The system call failed.

Action: Verify that 'date' executes correctly from the UNIX prompt.

Warning: at least one integer truncated to 32 bits
Warning: at least one integer truncated to 16 bits
Warning: at least one integer truncated to 8 bits

Cause: The number entered was too large for the currently specified display or access size.

Action: Try entering the command again using the correct size of number.

Width not in range: 1 <= valid width <= 80

Cause: You tried to specify the width of the field outside the range from 1 to 80.

Action: Try entering the command again using a valid number for the width.

Graphical/Softkey Interface Messages - Numbered

These numbered messages can occur because of various problems with the emulator/analyzer.

10315 **Logical emulator name unknown; not found in 64700tab file**

Cause: This message may occur while trying to start up the emulator. It indicates that the emulator name specified could not be found in the 64700tab.net or /etc/hosts files.

Action: Specify the name in one of these files.

10326 **Emulator locked by another user**

Cause: This message occurs when you try to start an emulation interface, but your attempt failed because the emulator is being used by someone else.

Action: The current user must release the emulator.

10327 **Cannot lock emulator; failure in obtaining the accessid**
Cannot lock emulator; failure in <ERRNO MSG>

10328 **Cannot unlock emulator; emulator not locked**

Cause: You have issued a command to unlock an emulator that is not locked.

Action: The emulator is available now. You can start the interface.

10328 **Cannot unlock emulator; lock file missing**
10328 **Cannot unlock emulator; semaphore missing**

Cause: Lock semaphore missing.

Action: Verify existence and permissions of /usr/hp64000 directory. Cycle emulator power and use **emul700 -u <logical name>**.

10328 **Cannot unlock emulator; emulator in use by user: <USER NAME>**

Cause: The emulator is already in use by the named user.

Action: Current user must release the emulator.

10329 **Emulator locked by user: <USER NAME>**

Cause: You tried to start an emulator interface, but your attempt failed because the emulator is already in use by someone else.

Action: Current user must release the emulator.

10330 **Emulator locked by another user interface**

Cause: You tried to start an emulator interface, but your attempt failed because the emulator is already in use by someone else.

Action: Current user must release the emulator.

10331 **HP64700 I/O channel in use by emulator: <LOGICAL NAME>**

Cause: You tried to start an emulator interface, but your attempt failed because the emulator is already in use by someone else.

Action: Current user must release the emulator.

10332 **Cannot default emulator; already in use**

Cause: You tried to start an emulator interface, but your attempt failed because the emulator is already in use by someone else.

Action: Current user must release the emulator.

10350 **Cannot interpret emulator output**

Cause: There may be characters dropped in the information returned from the emulator.

Action: Ignore this message unless it becomes frequent. If it becomes frequent, you may have a fatal error; call your HP 64700 representative.

10351 **Exceeded maximum 64700 command line length**

Cause: Your command is longer than 240 characters.

Action: Shorten the command.

10352 **Incompatible with 64700 firmware version**

Cause: The installed interface firmware combination is incorrect or incompatible.

Action: Upgrade the interface software of product firmware.

10360 **Analyzer limitation; all range resources in use**
Analyzer limitation; all pattern resources in use
Analyzer limitation; all expression resources in use

Cause: Your trace specification would use more than the maximum number of resources available to the analyzer.

Action: Simplify the trace specification.

10371 **64700 command aborted**

Cause: User abort occurred due to emulator being monopolized by another command.

Action Don't issue an abort.

Terminal Interface Messages

This section contains descriptions of error messages that can occur while using the Terminal Interface. The error messages are listed in numerical order, and each description includes the cause of the error and the action you should take to remedy the situation.

The emulator can return messages to the display only when it is prompted to do so. Situations may occur where an error is generated as the result of some command, but the error message is not displayed until the next command (or a carriage return) is entered.

A maximum number of 8 error messages can be displayed at one time. If more than 8 errors are generated, only the last 8 are displayed.

Emulator Messages

20 **Attempt to change foreground monitor map term**

Cause: When configuring the emulator to use a foreground monitor, a memory range is automatically mapped for the monitor's use. You attempted to alter that term when mapping memory.

Action: Try using another memory range for the new map term. If you need to have the range used by the foreground monitor, then switch to a background monitor, delete the old foreground monitor map term, and add the new term. Now you can return to using a foreground monitor; remember you will need to reload the monitor code.

21 **Insufficient emulation memory**

Cause: You have attempted to map more emulation memory than is available.

Action: Reduce the amount of emulation memory that you are trying to map.

40 **Restricted to real time runs**

Cause: While the emulator is restricted to real-time execution, you have attempted to use a command that requires a temporary break in execution to the monitor. The emulator does not permit the command and issues this error message.

Action: You must break the emulator's execution into the monitor before you can enter the command.

61 **Emulator is in the reset state**

Cause: You have entered a command that requires the emulator to be running in the monitor (for example, displaying registers).

Action: Enter the **break** command to cause the emulator to run in the monitor, and enter the command that caused the error again.

80 **Stack pointer is odd**

Cause: You have attempted to modify the stack pointer to an odd value for a processor that expects the stack to be aligned on a word boundary (such as the 68340).

Action: Modify the stack pointer to an even value.

81 **Stack is in guarded memory**

Cause: Your stack pointer pointed to a location in memory mapped as guarded; you then attempted to run or step the emulation processor. The emulator was unable to access the stack to complete the transition from the monitor to the user program or vice versa.

Action: Either remap memory so the stack pointer points to a location in RAM, or change the stack pointer value (either with your program or by configuring the emulator's stack pointer value on reset) to point to a location in RAM.

82 **Stack is in target ROM**

Cause: Your stack pointer pointed to a location in memory mapped as target ROM; you then attempted to run or step the emulation processor. The emulator was unable to access the stack to complete the transition from the monitor to the user program or vice versa.

Action: Either remap memory so the stack pointer points to a location in RAM, or change the stack pointer value (either with your program or by configuring the emulator's stack pointer value on reset) to point to a location in RAM.

83 **Stack is in emulation ROM**

Cause: Your stack pointer pointed to a location in memory mapped as emulation ROM; you then attempted to run or step the emulation processor. The emulator was unable to access the stack to complete the transition from the monitor to the user program or vice versa.

Action: Either remap memory so the stack pointer points to a location in RAM, or change the stack pointer value (either with your program or by configuring the emulator's stack pointer value on reset) to point to a location in RAM.

84 **Program counter is odd**

Cause: You attempted to modify the program counter to an odd value using the **modify registers** command on a processor which expects even alignment of opcodes.

Action: Modify the program counter only to even numbered values.

102 **Monitor failure; no clock input**

Cause: The monitor is unable to run because no emulation processor clock is available.

Action: If running out of circuit, select the internal clock during configuration; if running in-circuit, select the external clock during configuration and make sure a clock meeting the microprocessor's specifications is input to the clock pin of the target system probe.

103 **Monitor failure; no processor cycles**

Cause: The monitor is unable to run since the processor is not running. The monitor is unable to determine the cause of the failure.

Action: If running in-circuit, troubleshoot the target system. If running out of circuit, reinitialize the emulator and try the procedure again.

104 **Monitor failure; bus grant**

Cause: The monitor is unable to run. The emulation processor is not running because it has granted the bus to another device.

Action: Wait until the processor has regained bus control, then retry the operation.

105 **Monitor failure; halted**

Cause: The monitor is unable to run because the processor is halted (due to an external halt line or a halt instruction).

Action: Release the external halt and retry the operation. If the processor halted due to a halt instruction, try the **reset** command, then retry the operation.

106 **Monitor failure; wait state**

Cause: The monitor is unable to run because the processor is in a continuous wait state.

Action: A continuous wait state may indicate target system problems. Troubleshoot the wait line. If you were running out of circuit, try initializing the emulator, then retry the procedure.

107 **Monitor failure; bus error**

Cause: The monitor is unable to run because the processor has encountered a bus fault (such as the 68340 /BERR line).

Action: Release the /BERR line and determine why it was activated.

68340 Emulator Messages

The following error messages are unique to the 68340 emulator.

141 **Dual ported memory limited to 4K bytes**

Cause: You attempted to map an address range larger than 4 Kbytes to dual-port emulation memory. Only 4 Kbytes of dual-port emulation memory is available.

Action: Split the address range into one 4 Kbyte range that will be mapped to dual-port emulation memory and another range that will be mapped to single-port emulation memory.

142 **Dual ported memory already in use**

Cause: You attempted to map a second address range to dual-port emulation memory. Only one address range can be mapped to dual-port emulation memory.

Action: If both ranges can fit into one 4 Kbyte range, delete the term currently mapped to dual-port emulation memory, and map the larger range. If both ranges

cannot fit into one 4 Kbyte range, you must map one of the ranges to single-port emulation memory.

143 **Dual ported memory in use by foreground monitor**

Cause: You attempted to map an address range to the 4 Kbyte block of dual-port emulation memory when it has already been allocated to the foreground monitor.

Action: When using a foreground monitor, only single-port emulation memory is available when mapping memory.

144 **Dual ported memory not mapped to <abs_file_address_range> for downloaded monitor**

Continuing with default foreground monitor

Cause: You attempted to load a custom foreground monitor absolute file whose address range does not agree with the range defined in response to the "Monitor address?" configuration question. The second message tells you the default foreground monitor, resident in the emulator firmware, continues to be used.

Action: Either assemble and link your foreground monitor at the address specified during configuration or change the answer to the "Monitor address?" configuration question so that it agrees with the monitor program absolute file.

145 **Downloaded monitor spans multiple 4K byte block boundaries**

Cause: You attempted to load a custom foreground monitor absolute file that does not fit into one 4 Kbyte address range.

Action: Edit your foreground monitor program source file so that it fits into one 4 Kbyte address range.

146 **Monitor must be mapped on a 4K byte boundary**

Cause: You attempted to define a foreground monitor base address that is not on a 4 Kbyte boundary.

Action: Specify the monitor program base address that is on a 4 Kbyte boundary (in other words, ending in 000H).

147 **cs0 can be used on one term only**

Cause: You attempted to map a second address range for global chip select operation. Only one address range can be mapped for this purpose.

Action: You must remove the current mapper term that is assigned the **cs0** attribute before you can map a different range for global chip select operation.

148 **Bus activity required to access dual ported memory**

Cause: In order for the system controller to be able to access dual-port emulation memory, the 68340 emulation processor's /AS signal must not remain active for too long a time (more than 100 microseconds, approximately). Typically, this situation occurs when the 68340 emulation processor attempts to access memory that is not present.

Action: Investigate the cause of the /AS signal being active too long.

149 **Register mbar valid bit not set**

Register cf_mbar valid bit not set

Cause: The contents of the MBAR or CF_MBAR register must be valid (that is, bit 0 of the register must be set to 1) before you can display of modify of the SIM or SIM configuration registers.

Action: Make sure the contents of the MBAR or CF_MBAR register are valid.

150 **Program counter is odd or uninitialized**

Program counter is located in guarded memory

Cause: You attempted to run or step from the current program counter when it contains an odd value or an address that is mapped as guarded memory.

Action: When running from the current program counter, make sure it contains an even address that is not in guarded memory.

151 **Supervisor stack pointer is odd or uninitialized**

Supervisor stack is located in emulation ROM

Supervisor stack is located in guarded memory

Cause: You are using a foreground monitor, and you have attempted to run or step when the stack pointer contains an odd value or when the stack pointer contains an address that is mapped as emulation ROM or guarded memory.

Action: Make sure the stack pointer contains an even address value that is not mapped as emulation ROM or guarded memory.

152 **Foreground monitor had unexpected exception %d**

Cause: The decimal number value of the vector offset (included with the message) tells you what type of exception caused the message. For example, if a priviledged opcode is attempted to be executed at the user access level, the decimal value 32 is shown with this error message.

Action: Determine the cause of the exception.

155 **Unable to verify trace vector; vector table in guarded memory**

Cause: In order to step when using a foreground monitor, the trace vector must contain the address of the monitor program's TRACE_ENTRY label (which equals the monitor base address plus 800H). The step command reads the trace vector on each step to make sure it contains the correct address value. This error occurs when the vector table is in guarded memory.

Action: Make sure the vector table is not in an address range mapped as guarded memory.

156 **Unable to verify trace vector; vector table read failed**

Cause: In order to step when using a foreground monitor, the trace vector must contain the address of the monitor program's TRACE_ENTRY label (which equals the monitor base address plus 800H). The step command reads the trace vector on each step to make sure it contains the correct address value. This error occurs when the vector table is in target memory and the read fails.

Action: Make sure the vector base register points to the correct location in target memory and that the memory system is operating correctly.

157 **Unable to set trace vector to <TRACE_ENTRY_address>; vector table write failed**

Cause: In order to step when using a foreground monitor, the trace vector must contain the address of the monitor program's TRACE_ENTRY label (which equals the monitor base address plus 800H). The step command reads the trace vector on each step to make sure it contains the correct address value. If the trace vector does not contain the correct value, the emulator attempts to write the correct value. This error occurs when the vector table is in target memory and the write fails.

Action: Make sure the vector base register points to the correct location in target memory and that the memory system is writeable and is operating correctly.

158 **Trace vector modified to <TRACE_ENTRY_address> for single stepping**

Cause: In order to step when using a foreground monitor, the trace vector must contain the address of the monitor program's TRACE_ENTRY label (which equals the monitor base address plus 800H). The step command reads the trace vector on each step to make sure it contains the correct address value. If the trace vector does not contain the correct value, the emulator attempts to write the correct value. This message informs you the emulator was successful in writing the correct value to the trace vector.

159 **Unable to set trace vector to <TRACE_ENTRY_address>; vector table in TROM**

Cause: In order to step when using a foreground monitor, the trace vector must contain the address of the monitor program's TRACE_ENTRY label (which equals the monitor base address plus 800H). The step command reads the trace vector on each step to make sure it contains the correct address value. If the trace vector does not contain the correct value, the emulator attempts to write the correct value. This error occurs when the vector table is in memory mapped as target ROM.

Action: If the vector table is really in target ROM memory, its trace vector must already contain the correct address value.

162 **Double bus fault occurred**

Cause: This message informs you of a double bus fault in the emulation microprocessor. The most common cause of this error message is running or stepping from target memory locations that do not exist.

163 **Update HP64740 firmware to version A.02.02 or newer**

Cause: The HP 64751 emulator requires version A.02.02 or newer of the analyzer firmware.

Action: Refer to the "Updating Emulator/Analyzer Firmware" section in the "Installation" chapter.

168 **Can’t access module regs, addr space mask (@sd bit 6) is set**

Cause: You have attempted to modify the MBAR or CF_MBAR register with a value that masks supervisor data space by setting bit 6. The HP 64751 emulator requires an address space to access 68340 SIM registers and, therefore, doesn’t let you set bit 6 in the MBAR or CF_MBAR register.

Action: Modify the MBAR or CF_MBAR registers with values that do not set bit 6.

168 **Can’t access module regs, mbar (@sd bit 6) is set**

Can’t access module regs, cf_mbar (@sd bit 6) is set

Cause: These errors occur if the user program sets bit 6 of the MBAR register (whose contents can be copied into the CF_MBAR register with the **sync_sim_registers to_m68340_sims** command) and the emulator needs to access other SIM or emulator configuration registers.

Action: Modify the MBAR or CF_MBAR registers with values that do not set bit 6.

177 **BDM communication failed**

Cause: There was some problem with the serial communication interface with the emulation processor’s Background Debug Mode (BDM).

Action: Initialize the emulator or cycle power. Then reenter the command. If the same failure occurs, call your HP sales and service office.

178 **Unable to run performance verification tests**

Cause: The emulator was unable to run the performance verification tests.

Action: Make sure the emulator probe is connected correctly and that all cables are secured. Make sure that the emulator probe is plugged into the demo target system and that the demo target system power cable is connected to the HP 64700.

179 **M68340 probe not connected or configured incorrectly**

Cause: The emulator is reading an invalid identifier for the emulation probe.

Action: Make sure that the emulator probe cables are connected correctly. Also, make sure that the probe is the 68340 probe.

179 **Fuse F1 blown on HP64748C ABG Control Board**

Cause: The emulator detects a blown fuse on the HP 64748C ABG Control Board.

Action: Contact your Hewlett-Packard Representative.

General Emulator and System Messages

204 **FATAL SYSTEM SOFTWARE ERROR**

205 **FATAL SYSTEM SOFTWARE ERROR**

208 **FATAL SYSTEM SOFTWARE ERROR**

Cause: The system has encountered an error from which it cannot recover.

Action: Write down the sequence of commands which caused the error. Cycle power on the emulator and reenter the commands. If the error repeats, call your local HP Sales and Service office for assistance.

206 **Incompatible compatibility table entry**

Cause: The emulation firmware (ROM) is not compatible with the analysis or system firmware in your HP 64700 system.

Action: The ROMs in your emulator must be compatible with each other for your emulation system to work correctly. Contact your Hewlett-Packard Representative.

312 **Ambiguous address: %s**

Cause: Certain emulators support segmentation or function code information in addressing. The emulator is unable to determine which of two or more address ranges you are referring to, based upon the information you entered.

Action: Re-enter the command and fully specify the address, including segmentation or function code information.

318 **Count out of bounds: %s**

Cause: You specified an occurrence count less than 1 or greater than 65535.

Action: Re-enter the command, specifying a count value from 1 to 65535.

400 **Record checksum failure**

Cause: During a **transfer** operation, the checksum specified in a file did not agree with that calculated by the HP 64700.

Action: Retry the **transfer** operation. If the failure is repeated, make sure that both your host and the HP 64700 data communications parameters are configured correctly.

401 **Records expected: %s; records received: %s**

Cause: The HP 64700 received a different number of records than it expected to receive during a **transfer** operation.

Action: Retry the **transfer**. If the failure is repeated, make sure that the data communications parameters are set correctly on the host and on the HP 64700. Refer to the "Installation" chapter for details.

410 **File transfer aborted**

Cause: A **transfer** operation was aborted due to a break received, most likely a <CTRL>c from the keyboard.

Action: If you typed <CTRL>c, you probably did so because you thought the transfer was about to fail. Retry the transfer, making sure to use the correct command options. If you are unsuccessful, make sure that the data communications parameters are set correctly on the host and on the HP 64700, then retry the operation.

411 **Severe error detected, file transfer failed**

Cause: An unrecoverable error occurred during a **transfer** operation.

Action: Retry the transfer. If it fails again, make sure that the data communications parameters are set correctly on the host and on the HP 64700. Also make sure that you are using the correct command options, both on the HP 64700 and on the host.

412 **Retry limit exceeded, transfer failed**

Cause: The limit for repeated attempts to send a record during a **transfer** operation was exceeded, therefore the transfer was aborted.

Action: Retry the transfer. Make sure you are using the correct command options for both the host and the HP 64700. The data communications parameters need to be set correctly for both devices. Also, if you are in a remote location from the host, it is possible that line noise may cause the failure.

413 **Transfer failed to start**

Cause: Communication link or transfer protocol incorrect.

Action: Check link and transfer options.

415 **Timeout, receiver failed to respond**

Cause: Communication link or transfer protocol incorrect.

Action: Check link and transfer options.

600 **Adjust PC failed during break**

Cause: System failure or target condition.

Action: Run performance verification (Terminal Interface **pv** command), and check target system.

602 **Break failed**

Cause: The **break** command was unable to break the emulator to the monitor.

Action: Determine why the break failed, then correct the condition and retry the command. See message 608.

603 **Read PC failed during break**

Cause: System failure or target condition.

Action: Try again.

604 **Disable breakpoint failed: %s**

Cause: System failure or target condition.

Action: Run performance verification (Terminal Interface **pv** command), and check target system.

605 **Undefined software breakpoint: %s**

Cause: The emulator has encountered a software breakpoint in your program that was not inserted with the **modify software_breakpoints set** command.

Action: Remove the breakpoint instructions in your code before assembly and link.

606 **Unable to run after CMB break**

Cause: System failure or target condition.

Action: Run performance verification (Terminal Interface **pv** command), and check target system.

608 **Unable to break**

Cause: This message is displayed if the emulator is unable to break to the monitor because the emulation processor is reset, halted, or is otherwise disabled.

Action: First, look at the emulation prompt and other status messages displayed to determine why the processor is stopped. If reset by the emulation controller, use the **break** command to break to the monitor. If reset by the emulation system, release that reset. If halted, try **reset** and **break** to get to the monitor. If there is a bus grant, wait for the requesting device to release the bus before retrying the command. If there is no clock input, perhaps your target system is faulty. It's also possible that you have configured the emulator to restrict to real time runs, which will prohibit temporary breaks to the monitor.

610 **Unable to run**

Cause: System failure or target condition.

Action: Run performance verification (Terminal Interface **pv** command), and check target system.

611 **Break caused by CMB not ready**

Cause: This status message is printed during coordinated measurements if the CMB READY line goes false. The emulator breaks to the monitor. When CMB

READY is false, it indicates that one or more of the instruments participating in the measurement is running in the monitor.

Action: None, information only.

612 **Write to ROM break**

Cause: This status message will be printed if you have enabled breaks on writes to ROM and the emulation processor attempted a write to a memory location mapped as ROM.

Action: None (except troubleshooting your program).

613 **Analyzer Break**

Cause: Status message.

614 **Guarded memory access break**

Cause: This message is displayed if the emulation processor attempts to read or write memory mapped as guarded.

Action: Troubleshoot your program; or, you may have mapped memory incorrectly.

615 **Software breakpoint: %s**

Cause: This status message will be displayed if a software breakpoint is encountered during a program run. The emulator is broken to the monitor. The string %s indicates the address where the breakpoint was encountered.

616 **BNC trigger break**

Cause: This status message will be displayed if you have configured the emulator to break on a BNC trigger signal and the BNC trigger line is activated during a program run. The emulator is broken to the monitor.

617 **CMB trigger break**

Cause: This status message will be displayed if you have configured the emulator to break on a CMB trigger signal and the CMB trigger line is activated during a program run. The emulator is broken to the monitor.

618 **trig1 break**

Cause: This status message will be displayed if you use the **break_on_trigger** syntax of the **trace** command and the analyzer has found the trigger condition while tracing a program run. The emulator is broken to the monitor.

619 **trig2 break**

Cause: This status message will be displayed if you have used the internal **trig2** line to connect the analyzer trigger output to the emulator break input and the analyzer has found the trigger condition. The emulator is broken to the monitor.

620 **Unexpected software breakpoint**

Cause: If you have enabled software breakpoints, this message is displayed if a software breakpoint instruction is encountered in your program that was not inserted by a **modify software_breakpoints set** command and is therefore not in the breakpoint table.

Action: Remove the breakpoint instructions in your code before assembly and link, and use the **modify software_breakpoints set** command to reinsert them after the program is loaded into memory.

621 **Unexpected step break**

Cause: System failure.

Action: Run performance verification (Terminal Interface **pv** command).

622 **%s**

Cause: Monitor specific message.

623 **CMB execute break**

Cause: This message occurs when coordinated measurements are enabled and an EXECUTE pulse causes the emulator to run; the emulator must break before running.

Action: This is a status message; no action is required.

624 **Configuration aborted**

Cause: Occurs when a <CTRL>c is entered while emulator configuration items are being set.

626 **Configuration failed; setting unknown: %s=%s**

Cause: Target condition or system failure.

Action: Check target system, and run performance verification (Terminal Interface **pv** command).

628 **Guarded memory break: %s"**

Cause: A memory access to a location mapped as guarded memory has occurred during execution of the user program.

Action: Investigate the cause of the guarded memory access by the user program.

628 **Write to ROM break: %s"**

Cause: When the emulator is configured to break on writes to ROM, a memory write access to a location mapped as ROM has occurred during execution of the user program.

Action: Investigate the cause of the write to ROM by the user program. You can configure the emulator so that it does not break on writes to ROM.

630 **Register access aborted**

Cause: Occurs when a <CTRL>c is entered during register display.

631 **Unable to read registers in class: %s**

Cause: The emulator was unable to read the registers you requested.

Action: To resolve this, you must look at the other status messages displayed. Most likely, the emulator was unable to break to the monitor to perform the register read. See message 608.

632 **Unable to modify register: %s=%s**

Cause: The emulator was unable to modify the register you requested.

Action: To resolve this, you must look at the other status messages displayed. It's likely that emulator was unable to break to the monitor to perform the register modification. See message 608.

634 **Display register failed: %s**

Cause: The emulator was unable to display the register you requested.

Action: To resolve this, you must look at the other status messages displayed. It's likely that emulator was unable to break to the monitor to perform the register display. See message 608.

636 **Register not writable: %s**

Cause: This error occurs when you attempt to modify a read only register.

Action: If this error occurs, you cannot modify the contents of the register with the **modify register** command.

637 **Register class cannot be modified: %s**

Cause: You tried to modify a register class instead of an individual register.

Action: You can only modify individual registers. Refer to the **display registers** command description for a list of register names.

640 **Unable to reset**

Cause: Target condition or system failure.

Action: Check target system, and run performance verification (Terminal Interface **pv** command).

650 **Unable to configure break on write to ROM**

Cause: The emulator controller is unable to configure for breaks on writes to ROM, possibly because the emulator was left in an unknown state or because of a hardware failure.

Action: Initialize the emulator or cycle power. Then reenter the command. If the same failure occurs, call your HP sales and service office.

651 **Unable to configure break on software breakpoints**

Cause: The emulator controller cannot enable breakpoints, possibly because the emulator is in an unknown state or because of a hardware failure.

Action: Initialize the emulator or cycle power, then re-enter the command. If the same failure occurs, call your HP sales and service office.

653 **Break condition configuration aborted**

Cause: Occurs when <CTRL>c is entered during the configuration of break conditions.

661 **Software breakpoint break condition is disabled**

Cause: You have attempted to set or clear a software breakpoint when software breakpoints are disabled.

Action: You must enable software breakpoints before you can set them.

663 **Specified breakpoint not in list: %s**

Cause: You tried to clear a software breakpoint that was not previously set. The string %s prints the address of the breakpoint you attempted to clear.

Action: You must first set a software breakpoint before it can be cleared.

664 **Breakpoint list full; not added: %s**

Cause: The software breakpoint table is already reached the maximum of 32 breakpoints. The breakpoint you just requested, with address %s, was not inserted.

Action: Clear breakpoints that are no longer in use. Then, set the new breakpoint.

665 **Enable breakpoint failed: %s**

Cause: System failure or target condition.

Action: Check memory mapping and configuration questions.

666 **Disable breakpoint failed: %s**

Cause: System failure or target condition.

Action: Check memory mapping and configuration questions.

667 **Breakpoint code already exists: %s**

Cause: You attempted to insert a breakpoint; however, there was already a software breakpoint instruction at that location which was not already in the breakpoint table.

Action: Your program code is apparently using the same instructions as used by the software breakpoints feature. Remove the breakpoint instructions from your program code and use the **modify software_breakpoints set** command to insert them.

668 **Breakpoint not added: %s**

Cause: You tried to insert a breakpoint in a memory location which was not mapped or was mapped as guarded memory.

Action: Insert breakpoints only within memory ranges mapped to emulation or target RAM or ROM.

669 **Breakpoint remove aborted**

Cause: Occurs when <CTRL>c is entered when clearing a software breakpoint.

670 **Breakpoint enable aborted**

Cause: Occurs when <CTRL>c is entered when setting software breakpoints.

671 **Breakpoint disable aborted**

Cause: Occurs when <CTRL>c is entered when disabling software breakpoints.

680 **Stepping failed**

Cause: Stepping has failed for some reason.

Action: Usually, this error message will occur with other error messages. Refer to the descriptions of the accompanying error messages to find out more about why stepping failed.

684 **Failed to disable step mode**

Cause: System failure.

Action: Run performance verification (Terminal Interface **pv** command).

686 **Stepping aborted; number steps completed: %d**

Cause: This message is displayed if a break was received during a **step** command with a step count greater than zero. The break could have been due to any of the break conditions or a <CTRL>c break. The number of steps completed is displayed.

688 **Step display failed**

Cause: System failure or target condition.

Action: Check memory mapping and configuration questions.

689 **Break due to cause other than step**

Cause: An activity other than a **step** command caused the emulator to break. This could include any of the break conditions or a <CTRL>c break.

692 **Trace error during CMB execute**

Cause: System failure.

Action: Run performance verification (Terminal Interface **pv** command).

693 **CMB execute; run started**

Cause: This status message is displayed when you are making coordinated measurements. The CMB /EXECUTE pulse has been received; the emulation processor started running at the address specified by the **specify run** command.

Action: None; information only.

694 **Run failed during CMB execute**

Cause: System failure or target condition.

Action: Run performance verification (Terminal Interface **pv** command), and check target system.

700 **Target memory access failed**

Cause: This message is displayed if the emulator was unable to perform the requested operation on memory mapped to the target system.

Action: In most cases, the problem results from the emulator's inability to break to the monitor to perform the operation. See message 608.

702 **Emulation memory access failed**

Cause: System failure.

Action: Run performance verification (Terminal Interface **pv** command).

707 **Request access to guarded memory: %s**

Cause: The address or address range specified in the command included addresses within a range mapped as guarded memory. When the emulator attempts to access these during command processing, the above message is printed, along with the specific address or addresses accessed.

Action: Re-enter the command and specify only addresses or address ranges within emulation or target RAM or ROM. Or, you can remap memory so that the desired addresses are no longer mapped as guarded.

710 **Memory range overflow**

Cause: Accessing a word or short word, for example **display memory 0fffffff blocked word** will cause a rounding error that overflows physical memory.

Action: Reduce memory display request.

725 **Unable to load new memory map; old map reloaded**

Cause: There is not enough emulation memory left for this request.

Action: Reduce the amount of emulation memory requested.

726 **Unable to reload old memory map; hardware state unknown**

Cause: System failure.

Action: Run performance verification (Terminal Interface **pv** command).

754 **Memory modify aborted; next address: %s**

Cause: This message is displayed if a break occurs during processing of a **modify memory** command. The break could result from any of the break conditions (except a software breakpoint) or could have resulted from a <CTRL>c break.

Action: Retry the operation. If breaks are occurring continuously, you may wish to disable some of the break conditions.

901 **Invalid firmware for emulation subsystem**

Cause: This error occurs when the HP 64700 system controller determines that the emulation firmware (ROM) is invalid.

Action: This message is not likely to occur unless you have upgraded the ROMs in your emulator. Be sure that the correct ROM is installed in the emulation controller.

902 **Invalid analysis subsystem; product address: %s**

Cause: This error occurs when the HP 64700 system controller determines that the analysis firmware (ROM) is invalid.

Action: This message is not likely to occur unless you have upgraded the ROMs in your emulator. Be sure that the correct ROMs are installed in the analyzer board.

903 **Invalid ET subsystem; product address: %s**

Cause: Detects an invalid ET. Used only internally.

904 **Invalid auxiliary subsystem; product address: %s**

Cause: For future products.

911 **Lab firmware for emulation subsystem**

Cause: This message should never occur. It shows that you have an unreleased version of emulation firmware.

912 **Lab firmware analysis subsystem; product address: %s**

Cause: This message should never occur. It shows that you have an unreleased version of analysis firmware.

913 **Lab firmware subsystem; product address: %s**

Cause: This message should never occur. It shows that you have an unreleased version of system controller firmware.

914 **Lab firmware auxiliary subsystem; product address: %s**

Cause: This message should never occur. It shows that you have an unreleased firmware version of the auxiliary subsystem.

Analyzer Messages

1105 **Unable to delete label; used by emulation analyzer: <label>**

Cause: This error occurs when you attempt to delete an emulation trace label which is currently being used as a qualifier in the emulation trace specification or is currently specified in the emulation trace format.

Action: You stop the trace or must change the trace command before you can delete the label.

1108 **Unable to redefine label; used by emulation analyzer: <label>**

Cause: This error occurs when you attempt to redefine an emulation trace label which is currently used as a qualifier in the emulation trace specification.

Action: You stop the trace or must change the trace command before you can redefine the label.

1305 **CMB execute; emulation trace started**

Cause: This status message informs you that an emulation trace measurement has started as a result of a CMB execute signal (as specified by the **specify trace** command).

13

Specifications and Characteristics

Emulator Specifications and Characteristics

This section contains the following types of emulator specifications and characteristics:

- Electrical characteristics (including emulator timing).
- Physical characteristics.
- Environmental characteristics.

Electrical

This section describes the electrical characteristics of the HP 64751 68340 Emulator and the HP 64700 Card Cage.

Electrical Characteristics of the HP 64751 Emulator

The emulator requires some control signals and power supply in order to run. Therefore, a target system is required in order to use the emulator. The demo board that is included with the emulator is one such minimum target system.

DC Characteristics. The DC characteristics of the HP 64751 emulator's active probe are listed in the following table.

DC Electrical Specifications

Characteristic	Symbol	Min	Max	Unit
Input High Voltage (except clock)	V_{IH}	2.0	V_{CC}	V
Input Low Voltage	V_{IL}	GND	0.8	V
Input Leakage Current, GND≤V_{in}≤V_{CC} BR, BGACK, IRQx	I_{in}	 -2.5	 2.5	µA
Input High Current BERR, DSACKx RESET, HALT	I_{IH}	 — —	 25 50	µA
Input Low Current BERR, DSACKx RESET, HALT	I_{IL}	 — —	 -0.25 -1.0	mA
Output High Voltage, I_{OH} = -0.8 mA A0-A23, AS, BG, D0-D15, DS, R/W, RMC, SIZ0-SIZ1, FC0-FC3	V_{OH}	 2.4 2.4	 — —	V
Output Low Voltage I_{OL} = 2.0 mA A0-A23, SIZ0-SIZ1, FC0-FC3 I_{OL} = 4.5 mA R/W, RMC I_{OL} = 20 mA AS, D0-D15, DS I_{OL} = 14 mA RESET, HALT	V_{OL}	 — — — —	 0.5 0.5 0.5 0.5	V
Power Dissipation T_A = 0°C T_A = 70°C	P_D	 — —	 2.2 2.2	W
Capacitance, V_{in} = 0 V, T_A = 25°C, f = 1 MHz	C_{in}	—	20	pF
Load Capacitance A0-A31, R/W, SIZ0-SIZ1, FC0-FC3 All Other	C_L	 — —	 100 50	pF

AC Characteristics. The AC characteristics of the HP 64751 emulator's active probe are listed in the following tables.

AC Electrical Specifications (64751-66508 and higher active probe board numbers) — Clock Timing

Num.	Characteristic	Symbol	MC68340 25.16 MHz Min	MC68340 25.16 MHz Max	HP 64751 Min	HP 64751 Max	Unit
	System Frequency (See Note)	f_{sys}	dc	25.16	dc	25.16	MHz
	Crystal Frequency	f_{XTAL}	25	50	25	50	kHz
	On-Chip VCO System Frequency	f_{sys}	0.13	25.16	0.13	25.16	MHz
	On-Chip VCO Frequency Range	f_{VCO}	0.1	50.3	0.1	50.3	MHz
	Crystal Oscillator Startup Time	t_{rc}	—	20	—	20	ms
1	CLKOUT Period	t_{cyc}	40	—	40	—	ns
2,3	CLKOUT Pulse Width	t_{CW}	19	—	19	—	ns
4,5	CLKOUT Rise and Fall Times	t_{Crf}	—	4	—	4	ns
NOTE: All internal registers retain data at 0 Hz.							

AC Electrical Specifications (64751-66508 and higher active probe board numbers) — Read and Write Cycles
(Vcc = 5.0 Vdc +/-5%; GND = 0 Vdc; $T_A = T_L$ to T_H)

			MC68340 25.16 MHz		HP 64751		
Num.	**Characteristic**	**Symbol**	**Min**	**Max**	**Min**	**Max**	**Unit**
6	CLKOUT High to Address, FC, SIZE, $\overline{\text{RMC}}$ Valid	t_{CHAV}	0	20	0	20	ns
7	CLKOUT High to Address, Data, FC, SIZE, $\overline{\text{RMC}}$ High Impedance	t_{CHAZx}	0	40	0	40	ns
8	CLKOUT High to Address, FC, SIZE, $\overline{\text{RMC}}$ Invalid	t_{CHAZn}	0	—	0	—	ns
9	CLKOUT Low to $\overline{\text{AS}}$, $\overline{\text{DS}}$, $\overline{\text{CS}}$, $\overline{\text{IACK}}$, $\overline{\text{IFETCH}}$[A], $\overline{\text{IPIPE}}$[A] Asserted	t_{CLSA}	3 3 3	20 20 20	0 3 —	23 20 —	ns ns ns
9A[2]	$\overline{\text{AS}}$ to $\overline{\text{DS}}$ or $\overline{\text{CS}}$ Asserted (Read)	t_{STSA}	-6	6	-8	8	ns
11	Address, FC, SIZE, $\overline{\text{RMC}}$ Valid to $\overline{\text{AS}}$, $\overline{\text{CS}}$ (and $\overline{\text{DS}}$ Read) Asserted	t_{AVSA}	10	—	10	—	ns
12	CLKOUT Low to $\overline{\text{AS}}$, $\overline{\text{DS}}$, $\overline{\text{CS}}$, $\overline{\text{IACK}}$, $\overline{\text{IFETCH}}$[A], $\overline{\text{IPIPE}}$[A] Negated	t_{CLSN}	3 3 3	20 20 20	0 3 —	23 20 —	ns ns ns
13	$\overline{\text{AS}}$, $\overline{\text{DS}}$, $\overline{\text{CS}}$, $\overline{\text{IACK}}$ Negated to Address, FC, SIZE Invalid (Address Hold)	t_{SNAI}	10	—	10	—	ns
14	$\overline{\text{AS}}$, $\overline{\text{CS}}$ (and $\overline{\text{DS}}$ Read) Width Asserted	t_{SWA}	70	—	70	—	ns
14A	$\overline{\text{DS}}$ Width Asserted Write	t_{SWAW}	30	—	30	—	ns
14B	$\overline{\text{AS}}$, $\overline{\text{CS}}$, $\overline{\text{IACK}}$, (and $\overline{\text{DS}}$ READ) Width Asserted (Sync Cycle)	t_{SWDW}	30	—	30	—	ns
15[3]	$\overline{\text{AS}}$, $\overline{\text{DS}}$, $\overline{\text{CS}}$ Width Negated	t_{SN}	30	—	30	—	ns

AC Electrical Specifications (64751-66508 and higher active probe board numbers) — Read and Write Cycles
(Vcc = 5.0 Vdc +/-5%; GND = 0 Vdc; $T_A = T_L$ to T_H)

Num.	Characteristic	Symbol	MC68340 25.16 MHz		HP 64751		Unit
			Min	Max	Min	Max	
16	CLKOUT High to $\overline{AS}$, $\overline{DS}$, R/W High Impedance	t_{CHSZ}	— —	60 60	— —	$0.5t_{cyc}+30$ 40	ns ns
17	$\overline{AS}$, $\overline{DS}$, $\overline{CS}$ Negated to R/$\overline{W}$ High	t_{SNRN}	10	—	10	—	ns
18	CLKOUT High to R/$\overline{W}$ High	t_{CHRH}	0	20	0	20	ns
20	CLKOUT High to R/$\overline{W}$ Low	t_{CHRL}	0	20	0	20	ns
21	R/$\overline{W}$ High to $\overline{AS}$, $\overline{CS}$ Asserted	t_{RAAA}	10	—	10	—	ns
22	R/$\overline{W}$ Low to $\overline{DS}$ Asserted (Write)	t_{RASA}	47	—	47	—	ns
23	CLKOUT High to Data-Out Valid	t_{CHDO}	—	20	—	23	ns
24	Data-Out Valid to Negating Edge of $\overline{AS}$, $\overline{CS}$ (Synchronous Write)	t_{DVASN}	10	—	10	—	ns
25	$\overline{DS}$, $\overline{CS}$ Negated to Data-Out Invalid (Data-Out Hold)	t_{SNDOI}	10	—	10	—	ns
26	Data-Out Valid to $\overline{DS}$ Asserted (Write)	t_{DVSA}	10	—	9	—	ns
27	Data-In Valid to CLKOUT Low (Data Setup)	t_{DICL}	5	—	8	—	ns
27A	Late $\overline{BERR}$, $\overline{HALT}$, $\overline{BKPT}$[B] Asserted to CLKOUT Low (Setup Time)	t_{BELCL}	10 10	— —	10 —	— —	ns ns
28	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{DSACKx}$, $\overline{BERR}$, $\overline{HALT}$	t_{SNDN}	0	50	0	50	ns
29[4]	$\overline{DS}$ Negated to Data-In Invalid (Data-In Hold)	t_{SNDI}	0	—	0	—	ns
29A[4]	$\overline{DS}$ Negated to Data-In High Impedance	t_{SHDI}	—	40	—	40	ns

AC Electrical Specifications (64751-66508 and higher active probe board numbers) — Read and Write Cycles (Vcc = 5.0 Vdc +/-5%; GND = 0 Vdc; $T_A = T_L$ to T_H)

Num.	Characteristic	Symbol	MC68340 25.16 MHz Min	MC68340 25.16 MHz Max	HP 64751 Min	HP 64751 Max	Unit
30[4]	CLKOUT Low to Data-In Invalid (Synchronous Hold)	t_{CLDI}	10	—	10	—	ns
30A[4]	CLKOUT Low to Data-In High Impedance	t_{CLDH}	—	60	—	60	ns
31[5]	$\overline{\text{DSACKx}}$ Asserted to Data-In Valid	t_{DADI}	—	32	—	32	ns
32	$\overline{\text{HALT}}$ and $\overline{\text{RESET}}$ Input Transition Time	t_{RHrf}	0	140	0	140	ns
33	CLKOUT Low to $\overline{\text{BG}}$ Asserted	t_{CLBA}	—	20	—	20	ns
34	CLKOUT Low to $\overline{\text{BG}}$ Negated	t_{CLBN}	—	20	—	20	ns
35[7]	$\overline{\text{BR}}$ Asserted to $\overline{\text{BG}}$ Asserted ($\overline{\text{RMC}}$ Not Asserted)	t_{BRAGA}	1	—	1	—	clks
37	$\overline{\text{BGACK}}$ Asserted to $\overline{\text{BG}}$ Negated	t_{GAGN}	1	2.5	1	2.5	clks
39	$\overline{\text{BG}}$ Width Negated	t_{GH}	2	—	2	—	clks
39A	$\overline{\text{BG}}$ Width Asserted	t_{GA}	1	—	1	—	clks
46	R/$\overline{\text{W}}$ Width Asserted (Write or Read)	t_{RWA}	100	—	100	—	ns
46A	R/$\overline{\text{W}}$ Width Asserted (Sync. Write or Read)	t_{RWAS}	60	—	60	—	ns
47A	Asynchronous Input Setup Time	t_{AIST}	5	—	5	—	ns
47B	Asynchronous Input Hold Time	t_{AIHT}	10	—	10	—	ns
48[8]	$\overline{\text{DSACKx}}$ Asserted to $\overline{\text{BERR}}$, $\overline{\text{HALT}}$ Asserted	t_{DABA}	—	20	—	20	ns
53	Data-Out Hold from CLKOUT High	t_{DOCH}	0	—	0	—	ns
54	CLKOUT High to Data-Out High Impedance	t_{CHDH}	—	20	—	20	ns
55	R/$\overline{\text{W}}$ Asserted to Data Bus Impedance Change	t_{RADC}	25	—	25	—	ns

AC Electrical Specifications (64751-66508 and higher active probe board numbers) — Read and Write Cycles
(Vcc = 5.0 Vdc +/-5%; GND = 0 Vdc; $T_A = T_L$ to T_H)

Num.	Characteristic	Symbol	MC68340 25.16 MHz		HP 64751		Unit
			Min	Max	Min	Max	
56	$\overline{\text{RESET}}$ Pulse Width (Reset Instruction)	t_{HRPW}	512	—	512	—	clks
57	$\overline{\text{BERR}}$ Negated to $\overline{\text{HALT}}$ Negated (Rerun)	t_{BNHN}	0	—	0	—	ns
70	CLKOUT Low to Data Bus Driven (Show Cycle)	t_{SCLDD}	0	30	—[C]	—[C]	ns
71	Data Setup Time to CLKOUT Low (Show Cycle)	t_{SCLDS}	10	—	—[C]	—	ns
72	Data Hold from CLKOUT Low (Show Cycle)	t_{SCLDH}	6	—	—[C]	—	ns

MC68340 NOTES:
1. All AC timing is shown with respect to 0.8-V and 2.0-V levels unless otherwise noted.
2. This number can be reduced to 5 ns if strobes have equal loads.
3. If multiple chip selects are used, the CS width negated (#15) applies to the time from the negation of a heavily loaded chip select to the assertion of a lightly loaded chip select.
4. These hold times are specified with respect to DS on asychronous reads and with respect to CLKOUT on synchronous reads. The user is free to use either hold time.
5. If the asychronous setup time (#47) requirements are satisfied, the DSACKx low to data setup time (#31) and DSACKx low to BERR low setup time (#48) can be ignored. The data must only satisfy the data-in to CLKOUT low setup time (#27) for the following clock cycle, BERR must only satisfy the late BERR low to CLKOUT low setup time (#27A) for the following clock cycle.
6. To ensure coherency during every operand transfer, BG will not be asserted in response to BR until after cycles of the current operand transfer are complete and RMC is negated.
7. In the absence of DSACKx, BERR is an asychronous input using the asychronous setup time (#47).
8. Address Access Time = $2t_{cyc} + t_{CW} - t_{CHAV} - t_{DICL}$ = 74 ns (@ 25.16-MHz clock). Chip Select Access Time = $2t_{cyc} - t_{CLSA} - t_{DICL}$ = 55 ns (@ 25.16-MHz clock).

HP 64751 NOTES:
A. IFETCH and IPIPE are not driven to the target system.
B. The emulator does not respond to BKPT from the target system.
C. The emulator does not drive data to the target system during show cycles.

AC Electrical Specifications (64751-66506 and lower active probe board numbers) — Clock Timing

Num.	Characteristic	Symbol	MC68340 16.78 MHz Min	MC68340 16.78 MHz Max	HP 64751 Min	HP 64751 Max	Unit
	System Frequency (See Note)	f_{sys}	dc	16.78	dc	16.78	MHz
	Crystal Frequency	f_{XTAL}	25	50	25	50	kHz
	On-Chip VCO System Frequency	f_{sys}	0.13	16.78	0.13	16.78	MHz
	On-Chip VCO Frequency Range	f_{VCO}	0.1	35	0.1	35	MHz
	Crystal Oscillator Startup Time	t_{rc}	—	100	—	100	ms
1	CLKOUT Period	t_{cyc}	59.6	—	59.6	—	ns
2,3	CLKOUT Pulse Width	t_{CW}	28	—	28	—	ns
4,5	CLKOUT Rise and Fall Times	t_{Crf}	—	5	—	5	ns
NOTE: All internal registers retain data at 0 Hz.							

AC Electrical Specifications (64751-66506 and lower active probe board numbers) — Read and Write Cycles
(Vcc = 5.0 Vdc +/-5%; GND = 0 Vdc; $T_A = T_L$ to T_H)

Num.	Characteristic	Symbol	MC68340 16.78 MHz Min	MC68340 16.78 MHz Max	HP 64751 Min	HP 64751 Max	Unit
6	CLKOUT High to Address, FC, SIZE, $\overline{RMC}$ Valid	t_{CHAV}	0	30	0	30	ns
7	CLKOUT High to Address, Data, FC, SIZE, RMC High Impedance	t_{CHAZx}	0	60	0	60	ns
8	CLKOUT High to Address, FC, SIZE, $\overline{RMC}$ Invalid	t_{CHAZn}	0	—	0	—	ns
9	CLKOUT Low to $\overline{AS}$, $\overline{DS}$, $\overline{CS}$, $\overline{IACK}$, $\overline{IFETCH}$[A], $\overline{IPIPE}$[A] Asserted	t_{CLSA}	3 3 3	30 30 30	0 3 —	33 30 —	ns ns ns
9A[2]	$\overline{AS}$ to $\overline{DS}$ or $\overline{CS}$ Asserted (Read)	t_{STSA}	-15	15	-15	15	ns
11	Address, FC, SIZE, $\overline{RMC}$ Valid to $\overline{AS}$, $\overline{CS}$ (and DS Read) Asserted	t_{AVSA}	15	—	15	—	ns
12	CLKOUT Low to $\overline{AS}$, $\overline{DS}$, $\overline{CS}$, $\overline{IACK}$, $\overline{IFETCH}$[A], $\overline{IPIPE}$[A] Negated	t_{CLSN}	3 3 3	30 30 30	0 3 —	33 30 —	ns ns ns
13	$\overline{AS}$, $\overline{DS}$, $\overline{CS}$, $\overline{IACK}$ Negated to Address, FC, SIZE Invalid (Address Hold)	t_{SNAI}	15	—	15	—	ns
14	$\overline{AS}$, $\overline{CS}$ (and $\overline{DS}$ Read) Width Asserted	t_{SWA}	100	—	100	—	ns
14A	$\overline{DS}$ Width Asserted Write	t_{SWAW}	45	—	45	—	ns
14B	$\overline{AS}$, $\overline{CS}$, $\overline{IACK}$, (and $\overline{DS}$ READ) Width Asserted (Sync Cycle)	t_{SWDW}	40	—	40	—	ns
15[3]	$\overline{AS}$, $\overline{DS}$, $\overline{CS}$ Width Negated	t_{SN}	40	—	40	—	ns

AC Electrical Specifications (64751-66506 and lower active probe board numbers) — Read and Write Cycles
(Vcc = 5.0 Vdc +/-5%; GND = 0 Vdc; $T_A = T_L$ to T_H)

Num.	Characteristic	Symbol	MC68340 16.78 MHz Min	MC68340 16.78 MHz Max	HP 64751 Min	HP 64751 Max	Unit
16	CLKOUT High to $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$ High Impedance	t_{CHSZ}	— —	60 60	— —	$0.5t_{cyc}$+40 60	ns ns
17	$\overline{AS}$, $\overline{DS}$, $\overline{CS}$ Negated to R/$\overline{W}$ High	t_{SNRN}	15	—	15	—	ns
18	CLKOUT High to R/$\overline{W}$ High	t_{CHRH}	0	30	0	30	ns
20	CLKOUT High to R/$\overline{W}$ Low	t_{CHRL}	0	30	0	30	ns
21	R/$\overline{W}$ High to $\overline{AS}$, $\overline{CS}$ Asserted	t_{RAAA}	15	—	15	—	ns
22	R/$\overline{W}$ Low to $\overline{DS}$ Asserted (Write)	t_{RASA}	70	—	70	—	ns
23	CLKOUT High to Data-Out Valid	t_{CHDO}	—	30	—	33	ns
24	Data-Out Valid to Negating Edge of $\overline{AS}$, $\overline{CS}$ (Synchronous Write)	t_{DVASN}	15	—	15	—	ns
25	$\overline{DS}$, $\overline{CS}$ Negated to Data-Out Invalid (Data-Out Hold)	t_{SNDOI}	15	—	15	—	ns
26	Data-Out Valid to $\overline{DS}$ Asserted (Write)	t_{DVSA}	15	—	12	—	ns
27	Data-In Valid to CLKOUT Low (Data Setup)	t_{DICL}	5	—	8	—	ns
27A	Late $\overline{BERR}$, $\overline{HALT}$, $\overline{BKPT}$[B] Asserted to CLKOUT Low (Setup Time)	t_{BELCL}	20 20	— —	20 —	— —	ns ns
28	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{DSACKx}$, $\overline{BERR}$, $\overline{HALT}$	t_{SNDN}	0	80	0	80	ns
29[4]	$\overline{DS}$ Negated to Data-In Invalid (Data-In Hold)	t_{SNDI}	0	—	0	—	ns
29A[4]	$\overline{DS}$ Negated to Data-In High Impedance	t_{SHDI}	—	60	—	60	ns

AC Electrical Specifications (64751-66506 and lower active probe board numbers) — Read and Write Cycles
(Vcc = 5.0 Vdc +/-5%; GND = 0 Vdc; $T_A = T_L$ to T_H)

Num.	Characteristic	Symbol	MC68340 16.78 MHz Min	MC68340 16.78 MHz Max	HP 64751 Min	HP 64751 Max	Unit
30[4]	CLKOUT Low to Data-In Invalid (Synchronous Hold)	t_{CLDI}	15	—	15	—	ns
30A[4]	CLKOUT Low to Data-In High Impedance	t_{CLDH}	—	90	—	90	ns
31[5]	$\overline{\text{DSACKx}}$ Asserted to Data-In Valid	t_{DADI}	—	50	—	50	ns
32	$\overline{\text{HALT}}$ and $\overline{\text{RESET}}$ Input Transition Time	t_{RHrf}	0	200	0	200	ns
33	CLKOUT Low to $\overline{\text{BG}}$ Asserted	t_{CLBA}	—	30	—	30	ns
34	CLKOUT Low to $\overline{\text{BG}}$ Negated	t_{CLBN}	—	30	—	30	ns
35[7]	$\overline{\text{BR}}$ Asserted to $\overline{\text{BG}}$ Asserted ($\overline{\text{RMC}}$ Not Asserted)	t_{BRAGA}	1	—	1	—	clks
37	$\overline{\text{BGACK}}$ Asserted to $\overline{\text{BG}}$ Negated	t_{GAGN}	1	2.5	1	2	clks
39	$\overline{\text{BG}}$ Width Negated	t_{GH}	2	—	2	—	clks
39A	$\overline{\text{BG}}$ Width Asserted	t_{GA}	1	—	1	—	clks
46	R/$\overline{\text{W}}$ Width Asserted (Write or Read)	t_{RWA}	150	—	150	—	ns
46A	R/$\overline{\text{W}}$ Width Asserted (Sync. Write or Read)	t_{RWAS}	90	—	90	—	ns
47A	Asynchronous Input Setup Time	t_{AIST}	5	—	5	—	ns
47B	Asynchronous Input Hold Time	t_{AIHT}	15	—	15	—	ns
48[8]	$\overline{\text{DSACKx}}$ Asserted to $\overline{\text{BERR}}$, $\overline{\text{HALT}}$ Asserted	t_{DABA}	—	30	—	30	ns
53	Data-Out Hold from CLKOUT High	t_{DOCH}	0	—	0	—	ns
54	CLKOUT High to Data-Out High Impedance	t_{CHDH}	—	30	—	30	ns
55	R/$\overline{\text{W}}$ Asserted to Data Bus Impedance Change	t_{RADC}	40	—	40	—	ns

AC Electrical Specifications (64751-66506 and lower active probe board numbers) — Read and Write Cycles
(Vcc = 5.0 Vdc +/-5%; GND = 0 Vdc; $T_A = T_L$ to T_H)

Num.	Characteristic	Symbol	MC68340 16.78 MHz Min	MC68340 16.78 MHz Max	HP 64751 Min	HP 64751 Max	Unit
56	RESET Pulse Width (Reset Instruction)	t_{HRPW}	512	—	512	—	clks
57	BERR Negated to HALT Negated (Rerun)	t_{BNHN}	0	—	0	—	ns
70	CLKOUT Low to Data Bus Driven (Show Cycle)	t_{SCLDD}	0	30	—[C]	—[C]	ns
71	Data Setup Time to CLKOUT Low (Show Cycle)	t_{SCLDS}	15	—	—[C]	—	ns
72	Data Hold from CLKOUT Low (Show Cycle)	t_{SCLDH}	10	—	—[C]	—	ns

MC68340 NOTES:
1. All AC timing is shown with respect to 0.8-V and 2.0-V levels unless otherwise noted.
2. This number can be reduced to 5 ns if strobes have equal loads.
3. If multiple chip selects are used, the CS width negated (#15) applies to the time from the negation of a heavily loaded chip select to the assertion of a lightly loaded chip select.
4. These hold times are specified with respect to DS on asychronous reads and with respect to CLKOUT on synchronous reads. The user is free to use either hold time.
5. If the asychronous setup time (#47) requirements are satisfied, the DSACKx low to data setup time (#31) and DSACKx low to BERR low setup time (#48) can be ignored. The data must only satisfy the data-in to CLKOUT low setup time (#27) for the following clock cycle, BERR must only satisfy the late BERR low to CLKOUT low setup time (#27A) for the following clock cycle.
6. To ensure coherency during every operand transfer, BG will not be asserted in response to BR until after cycles of the current operand transfer are complete and RMC is negated.
7. In the absence of DSACKx, BERR is an asychronous input using the asychronous setup time (#47).
8. Address Access Time = $2t_{cyc} + t_{CW} - t_{CHAV} - t_{DICL}$ = 112.2 ns (@ 16.78-MHz clock). Chip Select Access Time = $2t_{cyc} - t_{CLSA} - t_{DICL}$ = 84.2 ns (@ 16.78-MHz clock).

HP 64751 NOTES:
A. IFETCH and IPIPE are not driven to the target system.
B. The emulator does not respond to BKPT from the target system.
C. The emulator does not drive data to the target system during show cycles.

Electrical Characteristics of the HP 64700

The electrical characteristics of the HP 64700 communication ports are as follows.

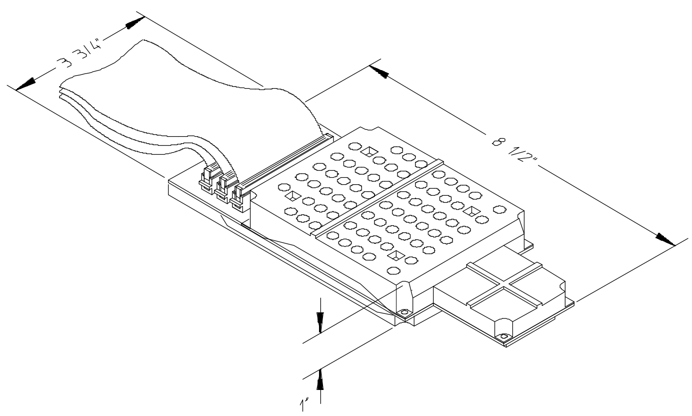
Communications

Serial Port	RS-232-C DCE or DTE to 38.4 Kbaud. RS-422 DCE to 460.8 Kbaud.
BNC (labeled TRIGGER IN/OUT)	Input. The signal must drive approximately 4 mA at 2 V. Edge sensitive. Minimum pulse width is approximately 25 ns. Output. Driven active high only; equals +2.4V into a 50 ohm load.

Physical

Dimensions of Emulator Probe

There must be enough clearance in the target system to allow the emulation probe to be plugged in and the cable routed from the target system to the emulator control card in the HP 64700. The following figure shows probe dimensions.



Emulator Dimensions

Width	325 mm (12.8 in.)
Height	173 mm (6.8 in.)
Length	389 mm (15.3 in.)

Emulator Weight

HP 64751	8.2 kg (18 lb)

Cable Length

Emulator to target system	approximately 914 mm (3 ft).

Probe Dimensions

92 mm (3.625 in.) width x 16 mm (0.626 in.) height x 159 mm (6.25 in.) length

Communications

Serial Port	25-pin female type "D" subminiature connector.
CMB Port	9-pin female type "D" subminiature connector.

CAUTION **Possible damage to emulator**. Any component used in suspending the emulator **must** be rated for 30 kg (65 lb) capacity.

Environmental

Temperature

Operating	0°C to +55°C (+32°F to 131°F)
Non-operating	-40°C to +70°C (-40°F to 158°F)

Altitude

Operating	4 600m (15 000 ft)
Non-operating	15 300m (50 000 ft).

Relative Humidity

15% to 95%.

Part 4

Concept Guide

Topics that explain concepts and apply them to advanced tasks.

Part 4

14

Concepts

Concepts

This chapter provides conceptual information on the following topics:

- X resources and the Graphical User Interface.

X Resources and the Graphical User Interface

This section contains more detailed information about X resources and scheme files that control the appearance and operation of the Graphical User Interface. This section:

- Describes the X Window concepts surrounding resource specification.
- Describes the Graphical User Interface's implementation of scheme files.

X Resource Specifications

An X resource specification is a resource name and a value. The resource name identifies the element whose appearance or behavior is to be defined, and the value specifies how the element should look or behave. For example, consider the following resource specification:

```
Application.form.row.done.background: red
```

The resource name is "Application.form.row.done.background:" and the value is "red".

Resource Names Follow Widget Hierarchy

A *widget* is an OSF/Motif graphic device from which X applications are built. For example, pushbuttons and menu bars are Motif widgets. Applications are built using a hierarchy of widgets, and the application's X resource names follow this hierarchy. For example:

```
Application.form.row.done.background: red
```

In the resource name above, the top-level widget is named after the application. One of the top-level widget's children is a form widget, one of the form widget's children is a row-column manager widget, and one of the row-column manager widget's children is a pushbutton widget. Resource names show a path in the widget hierarchy.

Each widget in the hierarchy is a member of a widget class, and the particular instance of the widget is named by the application programmer.

Class Names or Instance Names Can Be Used

When specifying resource names, you can use either instance names or class names. For example, a "Done" pushbutton may have an instance name of "done" and a class name of "XmPushButton". To set the background color for a hypothetical "Done" pushbutton, you can use:

```
Application.form.row.done.background: red
```

Or, you can use:

```
Application.form.row.XmPushButton.background: red
```

Applications also have class and instance names. For example, an application may have an instance name of "applic1" and a class name of "Application". To set the background color for a hypothetical "Done" pushbutton only in the "applic1" application, you can use:

```
applic1.form.row.done.background: red
```

Note that instance names are more specific than class names. That is, class names may apply to many instances of the widget.

The class and instance names for the widgets in the Graphical User Interface can be displayed by choosing **Help→X Resource Names** and clicking on the "All names" button.

Wildcards Can Be Used

A wildcard may be used to match a resource specification to many different widgets at once. For example, to set the background color of all pushbuttons, you can use:

```
Application*XmPushButton.background: red
```

Note that resource names with wildcards are more general than those without wildcards.

Specific Names Override General Names

A more specific resource specification will override a more general one when both apply to a particular widget or application.

The names for the application and the main window widget in HP64_Softkey applications have been chosen so that you may specify custom resource values that apply in particular situations:

1 Apply to ALL HP64_Softkey applications:

HP64_Softkey*<resource>: <value>

2 Apply to specific types of HP64_Softkey applications:

emul*<resource>: <value> (for the emulator)
perf*<resource>: <value> (for the performance analyzer)

3 Apply to all HP64_Softkey applications, but only when they are connected to a particular type of microprocessor:

m68340<resource>: <value> (for the 68340)
m68020<resource>: <value> (for the 68020)

4 Apply to a specific HP64_Softkey application connected to a specific processor:

perf.m68340*<resource>: <value> (for the 68340 perf. analyzer)
emul.m68020*<resource>: <value> (for the 68020 emulator)

If all four examples above are used for a particular resource, #3 will override #2 for all applications connected to a 68340 emulator, and #4 will override #2, but only for the specifically mentioned type of microprocessor.

When modifying resources, your resource paths must either match, or be more specific than, those found in the application defaults file.

How X Resource Specifications are Loaded

When the Graphical User Interface starts up, it loads resource specifications from a set of configuration files located in system directories as well as user-specific locations.

Application Default Resource Specifications

Default resource specifications for an application are placed in a system directory:

HP-UX	/usr/lib/X11/app-defaults
SunOS	/usr/openwin/lib/X11/app-defaults

The name of the Graphical User Interface application defaults file is HP64_Softkey (same as the application class name). This file is well-commented and contains information about each of the X resources you can modify. You can easily view this file by choosing **Help→Topic** and selecting the "X Resources: App Default File" topic. Do not modify the application defaults file; any changes to this file will affect the appearance and behavior of the application for all users.

User-Defined Resource Specifications

User-defined resources (for any X application) are located in the X server's RESOURCE_MANAGER property or in the user's $HOME/.Xdefaults file.

Load Order

Resource specifications are loaded from the following places in the following order:

5 The application defaults file. For example, /usr/lib/X11/app-defaults/HP64_Softkey when the operating system is HP-UX or /usr/openwin/lib/X11/app-defaults/HP64_Softkey when the operating system is SunOS.

6 The $XAPPLRESDIR/HP64_Softkey file. (The XAPPLRESDIR environment variable defines a directory containing system-wide custom application defaults.)

7 The server's RESOURCE_MANAGER property. (The **xrdb** command loads user-defined resource specifications into the RESOURCE_MANAGER property.)

   If no RESOURCE_MANAGER property exists, user defined resource settings are read from the $HOME/.Xdefaults file.

8 The file named by the XENVIRONMENT environment variable.

If the XENVIRONMENT variable is not set, the $HOME/.Xdefaults-*host* file is read (typically contains resource specifications for a specific remote host).

9 Resource specifications included in the command line with the **-xrm** option.

When specifications with identical resource names appear in different places, the latter specification overrides the former.

Scheme Files

Several of the Graphical User Interface's X resources identify *scheme files* that contain additional X resource specifications. Scheme files group resource specifications for different displays, computing environments, and languages.

Resources for Graphical User Interface Schemes

There are five X resources that identify scheme files:

HP64_Softkey.labelScheme:

Names the scheme file to use for labels and button text. Values can be: Label, $LANG, or a custom scheme file name. The default uses the $LANG environment variable if it is set and if a scheme file named Softkey.$LANG exists in one of the directories searched for scheme files; otherwise, the default is Label.

HP64_Softkey.platformScheme:

Names the subdirectory for the platform specific color, size, and input scheme files. This resource should be set to the platform on which the X server is running (and displaying the Graphical User Interface) if it is different than the platform where the application is running. Values can be: HP-UX, SunOS, pc-xview, or a custom platform scheme directory name.

HP64_Softkey.colorScheme:

Names the color scheme file. Values can be: Color, BW, or a custom scheme file name.

HP64_Softkey.sizeScheme:

Names the size scheme file which defines the fonts and the spacing used. Values can be: Large, Small, or a custom scheme file name.

HP64_Softkey.inputScheme:

Names the input scheme file which specifies mouse and keyboard operation. Values can be: Input, or a custom scheme file name.

The actual scheme file names take the form: "Softkey.<value>".

Scheme File Names

There are six scheme files provided with the Graphical User Interface. Their names and brief descriptions of the resources they contain follow.

Softkey.Label	Defines the labels for the fixed text in the interface. Such things as menu item labels and similar text are in this file. If the $LANG environment variable is set, the scheme file "Softkey.$LANG" is loaded if it exists; otherwise, the file "Softkey.Label" is loaded.
Softkey.BW	Defines the *color scheme* for black and white displays. This file is chosen if the display cannot produce at least 16 colors.
Softkey.Color	Defines the *color scheme* for color displays. This file is chosen if the display can produce 16 or more colors.
Softkey.Large	Defines the *size scheme* (that is, the window dimensions and fonts) for high resolution displays (1000 pixels or more vertically).
Softkey.Small	Defines the *size scheme* (that is, the window dimensions and fonts) for low resolution displays (less than 1000 pixels vertically).
Softkey.Input	Defines the *input scheme* (that is, the button and key bindings for the mouse and keyboard).

Load Order for Scheme Files

Scheme files are searched for in the following directories and in the following order:

10 System scheme files in directory /usr/hp64000/lib/X11/HP64_schemes.

11 System-wide custom scheme files located in directory $XAPPLRESDIR/HP64_schemes.

12 User-defined scheme files located in directory $HOME/.HP64_schemes (note the dot in the directory name).

Custom Scheme Files

You can modify scheme files by copying them to the directory for user-defined schemes and changing the resource specifications in the file. For example, if you wish to modify the color scheme, and your platform is HP-UX, you can copy the /usr/hp64000/lib/X11/HP64_schemes/HP-UX/Softkey.Color file to $HOME/.HP64_schemes/HP-UX/Softkey.Color and modify its resource specifications.

You can create custom scheme files by modifying the X resource for the particular scheme and by placing the custom scheme file in the directory for user-defined schemes. For example, if the following resource specifications are made:

```
HP64_Softkey.platformScheme:  HP-UX
HP64_Softkey.colorScheme:     MyColor
```

The custom scheme file would be:

```
$HOME/.HP64_schemes/HP-UX/Softkey.MyColor
```

Part 5

Installation Guide

Instructions for installing and configuring the product.

Part 5

15

Installation

Installation

This chapter shows you how to install emulation and analysis hardware and interface software. It also shows you how to verify installation by starting the emulator analyzer interface for the first time. These installation tasks are described in the following sections:

- Installing hardware.
- Connecting the HP 64700 to a computer or LAN.
- Installing HP 9000 software.
- Installing Sun SPARCsystem software.
- Verifying the installation.

Minimum HP 9000 Hardware and System Requirements

The following is a set of minimum hardware and system recommendations for operation of the Graphical User Interface on HP 9000 Series 300/400 and Series 700 workstations.

HP-UX For Series 9000/300 and Series 9000/400 workstations, the minimum supported version of the operating system is 7.03 or later. For Series 9000/700 workstations, the minimum supported version of the operating system is version 8.01.

Motif/OSF For Series 9000/700 workstations, you must also have the Motif 1.1 dynamic link libraries installed. They are installed by default, so you do not have to install them specifically for this product, but you should consult your HP-UX documentation for confirmation and more information.

Hardware and Memory Any workstation used with the Graphical User Interface should have a minimum of 16 megabytes of memory. Series 300 workstations should have a minimum performance equivalent to that of a HP 9000/350. A color display is also highly recommended.

From here, you should proceed to the section titled "Installation for HP 9000 Hosted Systems" for instructions on how to install, verify, and start the Graphical User Interface on HP 9000 systems.

Minimum Sun SPARCsystem Hardware and System Requirements

The following is a set of minimum hardware and system recommendations for operation of the Graphical User Interface on Sun SPARCsystem workstations.

SunOS The Graphical User Interface software is designed to run on a Sun SPARCsystem with SunOS version 4.1 or 4.1.1 or greater. The tape uses the QIC-24 data format.

64700 Operating Environment The Graphical User Interface requires version A.04.10 or greater of the 64700 Operating Environment. (The Graphical User Interface version is A.04.00.)

Hardware and Memory Any workstation used with the Graphical User Interface should have a minimum of 16 megabytes of memory. A color display is also highly recommended.

From here, you should proceed to the section titled "Installation for Sun SPARCsystems" for instructions on how to install, verify, and start the Graphical User Interface on SPARCsystem workstations.

Installing Hardware

This section describes how to install emulation and analysis hardware and how to connect the emulator probe to the demo target system.

Equipment supplied

The minimum system contains:

- HP 64751A 68340 PGA Emulator Probe (which includes the demo target system).
- HP 64748C Emulation Control card.
- HP 64704A 80-Channel Emulation Bus Analyzer card.
- HP 64700 Card Cage.

Optional parts are:

- HP 64171A 256 Kbyte Memory Modules or HP 64171B 1 Mbyte Memory Modules (0 wait state emulation memory through 16.7 MHz, 1 wait state above 16.7 MHz).
- HP 64172A 256 Kbyte Memory Modules or HP 64172B 1 Mbyte Memory Modules (0 wait state emulation memory through 25 MHz).
- HP 64173A 4 Mbyte Memory Modules (0 wait state emulation memory through 22 MHz, 1 wait state above 22 MHz).

Equipment and tools needed

In order to install and use the 68340 emulation system, you need:

- Flat-blade screwdriver.

Installation overview

The steps in the installation process are:

1. Connect the HP 64751 emulator probe to the HP 64748C emulator control card.
2. Install cards into the HP 64700 card cage.
3. Install emulation memory modules on the emulator probe.
4. Connect the emulator probe to the demo target system.
5. Apply power to the HP 64700.

Your emulation and analysis system may already be assembled (depending on how parts of the system were ordered).

Antistatic precautions

Integrated-circuit boards contain electrical components that are easily damaged by small amounts of static electricity. To avoid damage to the emulator cards, follow these guidelines:

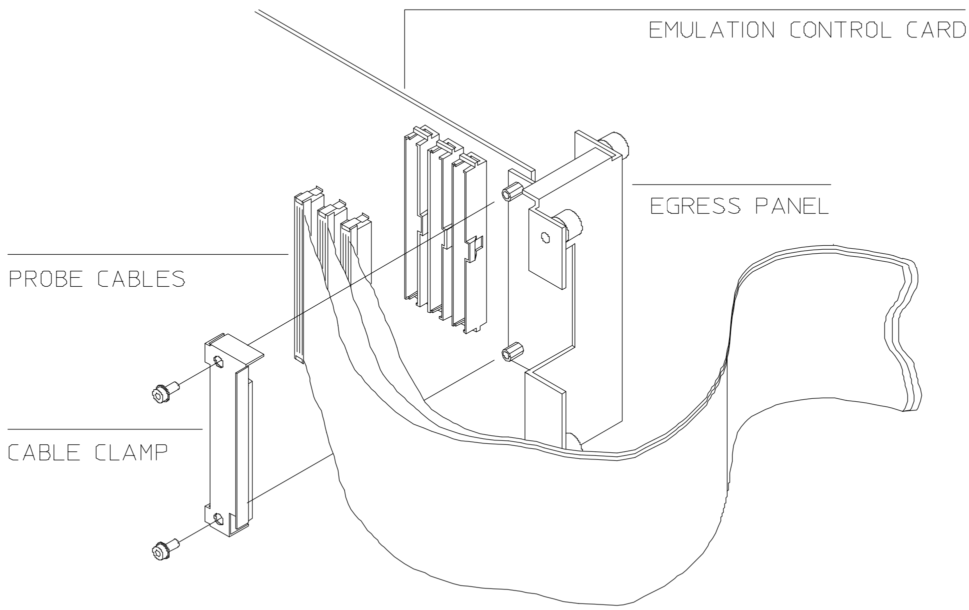
- If possible, work at a static-free workstation.
- Handle the boards only by the edges; do not touch components or traces.
- Use a grounding wrist strap that is connected to the HP 64700's chassis.

Step 1. Connect the Emulator Probe Cables

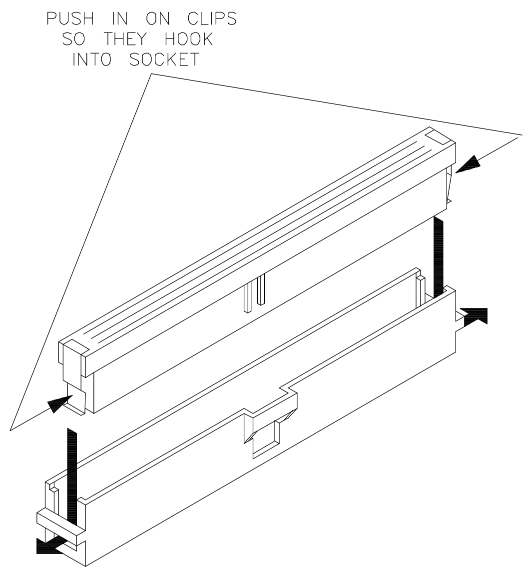
Three ribbon cables connect the HP 64748C emulation control card to the HP 64751 68340 emulator probe.

The shortest cable connects from J1 of the emulation control card to J3 of the emulator probe. The medium length cable connects from J2 of the emulation control card to J2 of the emulator probe. The longest cable connects from J3 of the emulation control card to J1 of the emulator probe.

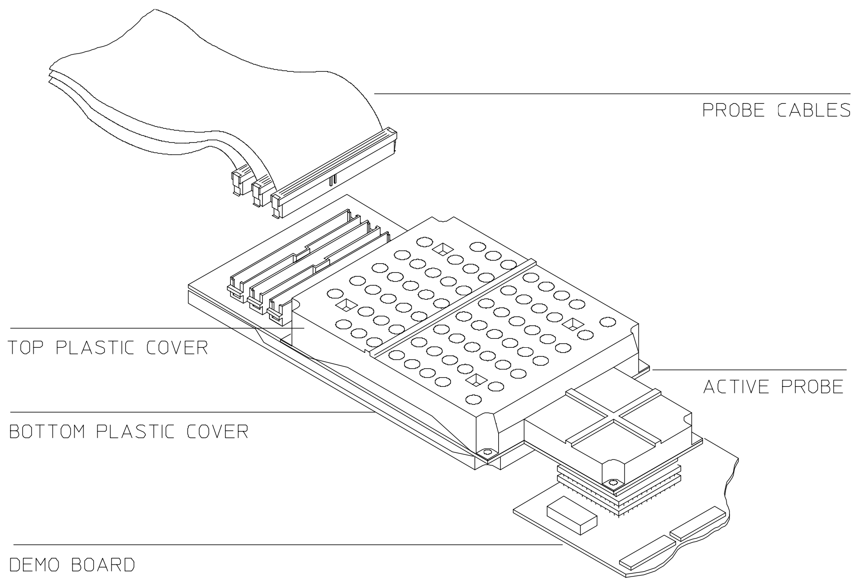
1 Connect the emulator probe cables to the emulation control card.



2 When inserting cable connectors into the sockets, press inward on the connector clips so that they hook into the sockets as shown.



3 Connect the other ends of the cables to the emulator probe.

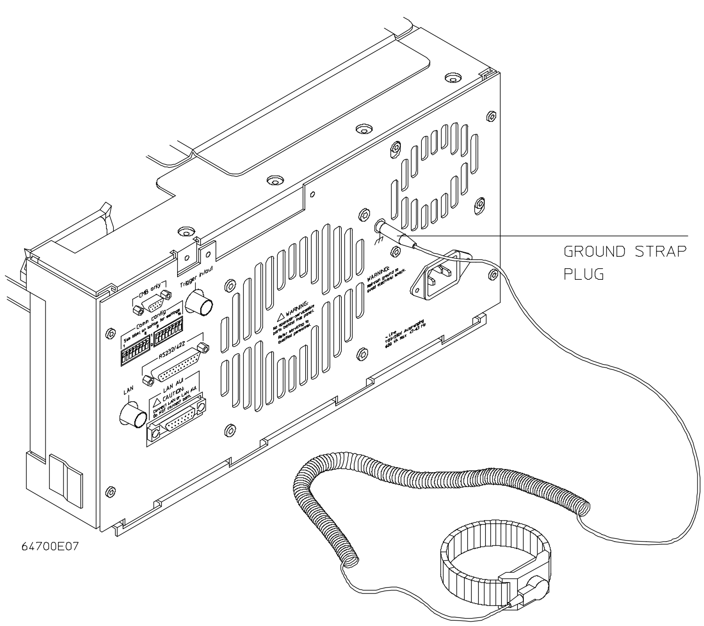


Step 2. Install Boards into the HP 64700 Card Cage

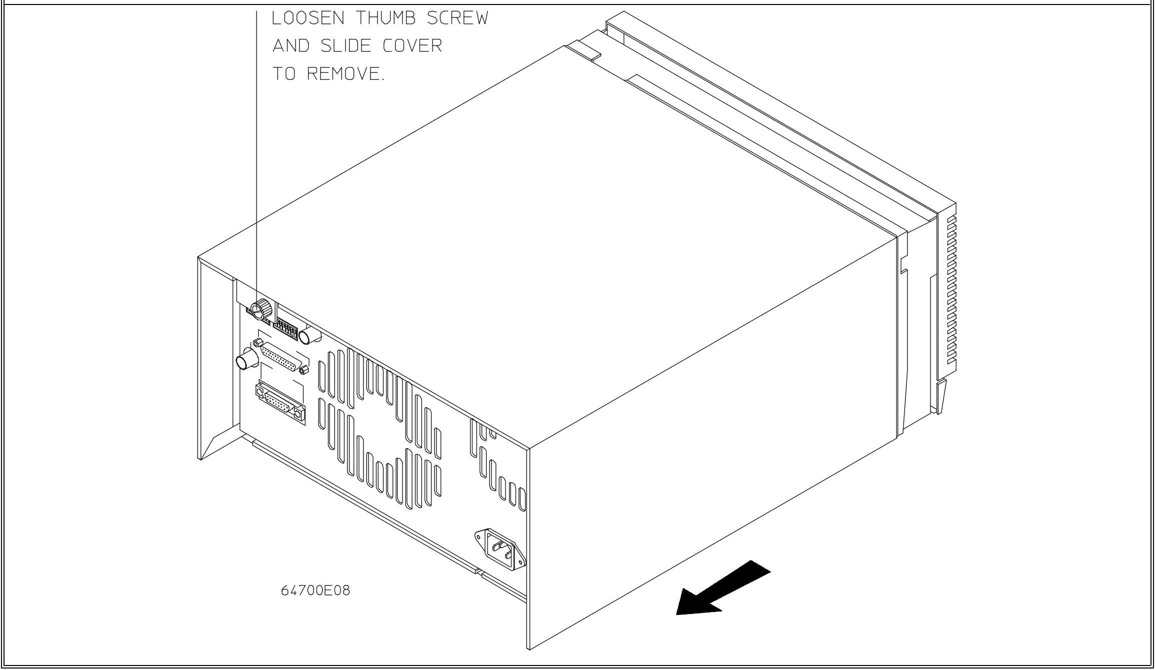
WARNING **Before removing or installing parts in the HP 64700 Card Cage, make sure that the card cage power is off and that the power cord is disconnected.**

CAUTION Do NOT stand the HP 64700 on the rear panel. You could damage the rear panel ports and connectors.

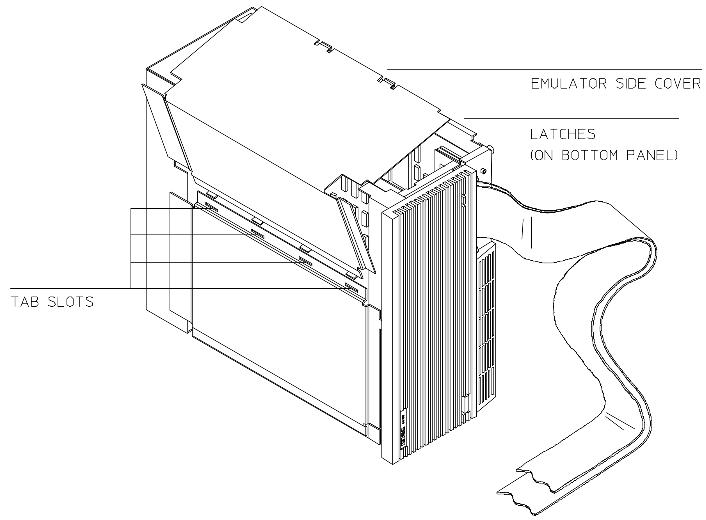
1 Use a ground strap when removing or installing boards into the HP 64700 Card Cage to reduce the chances of damage to the circuit cards from static discharge. A jack on the rear panel of the HP 64700 Card Cage is provided for this purpose.



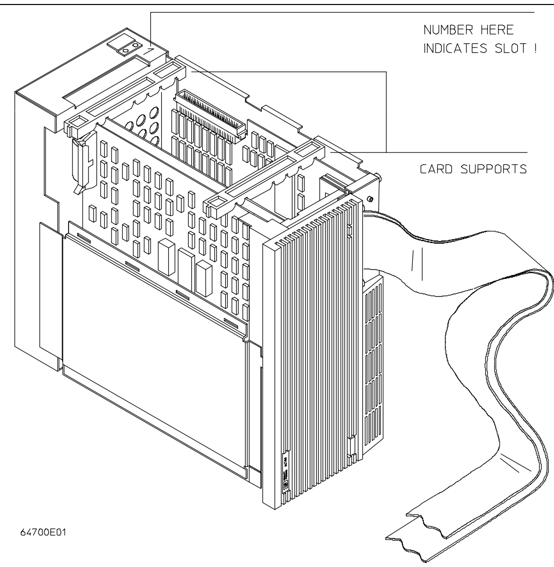
2 Turn the thumb screw and remove the top cover by sliding the cover toward the rear and up.



3 Remove the side cover by unsnapping the two latches and lifting off.

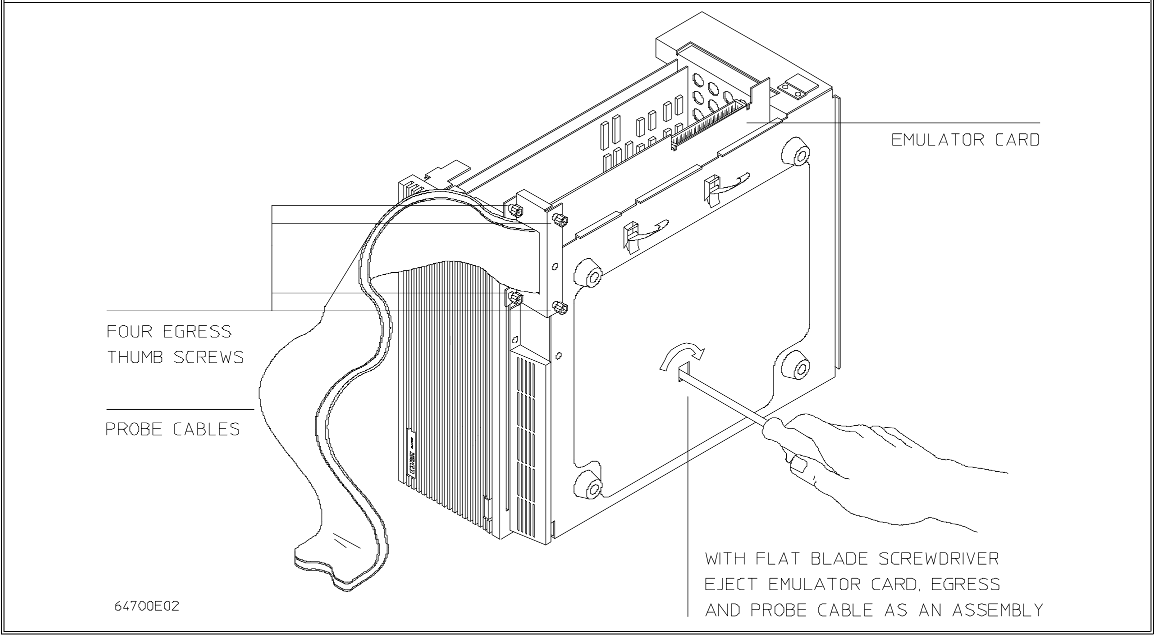


4 Remove the card supports.

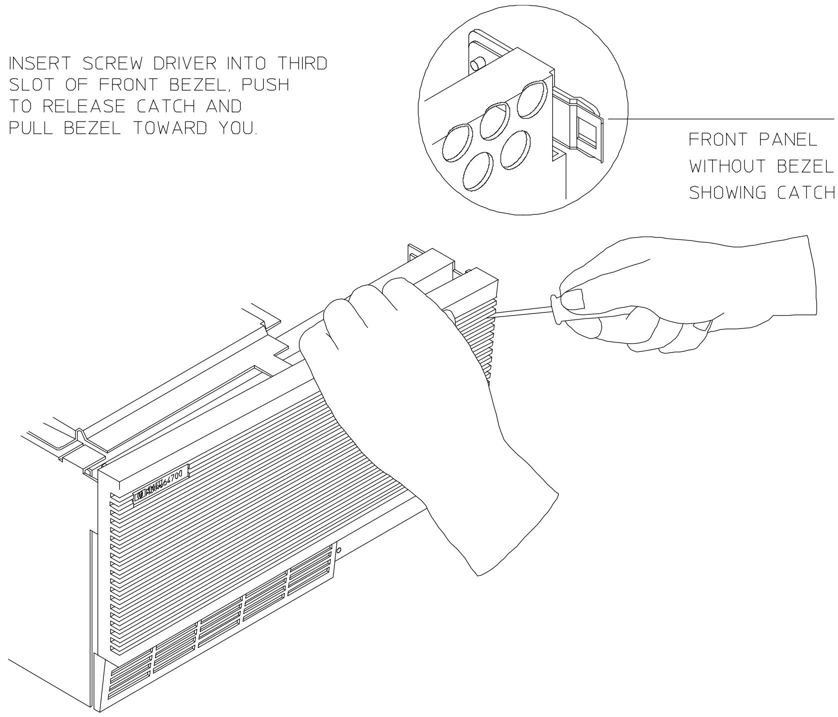


5 First, completely loosen the four egress thumb screws.

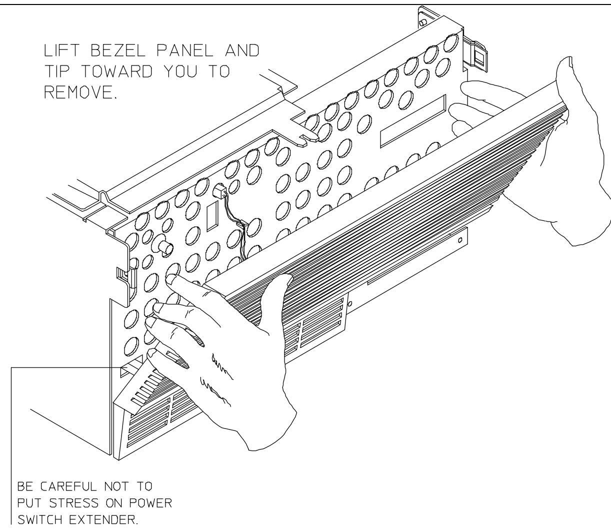
To remove emulator cards, insert a flat blade screwdriver in the access hole and eject the emulator cards by rotating the screwdriver.



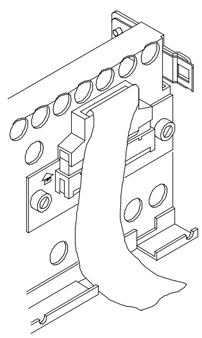
6 Insert a screw driver into the third slot of the right side of the front bezel, push to release catch, and pull the right side of the bezel about one half inch away from the front of the HP 64700. Then, do the same thing on the left side of the bezel. When both sides are released, pull the bezel toward you approximately 2 inches.



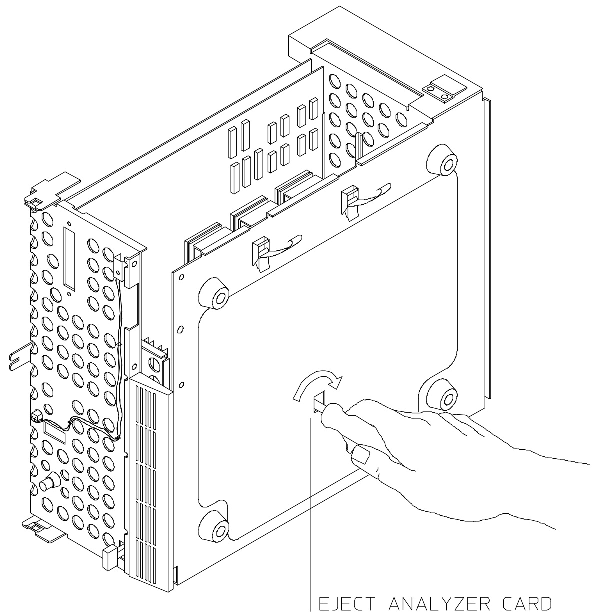
7 Lift the bezel panel to remove. Be careful not to put stress on the power switch extender.



8 If you're removing an existing analyzer card that provides external analysis, remove the right angle adapter board by turning the thumb screws counter-clockwise.



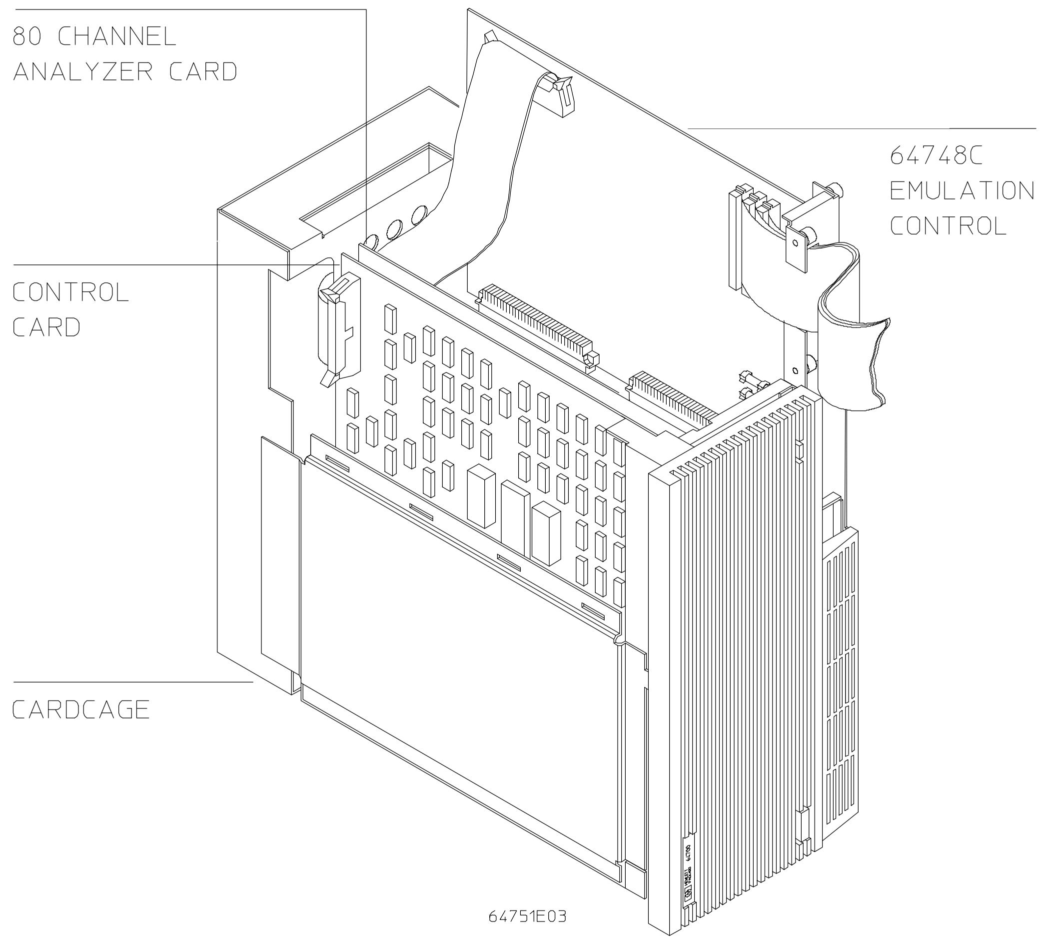
9 To remove the analyzer card, insert a flat blade screwdriver in the access hole and eject the analyzer card by rotating the screwdriver.



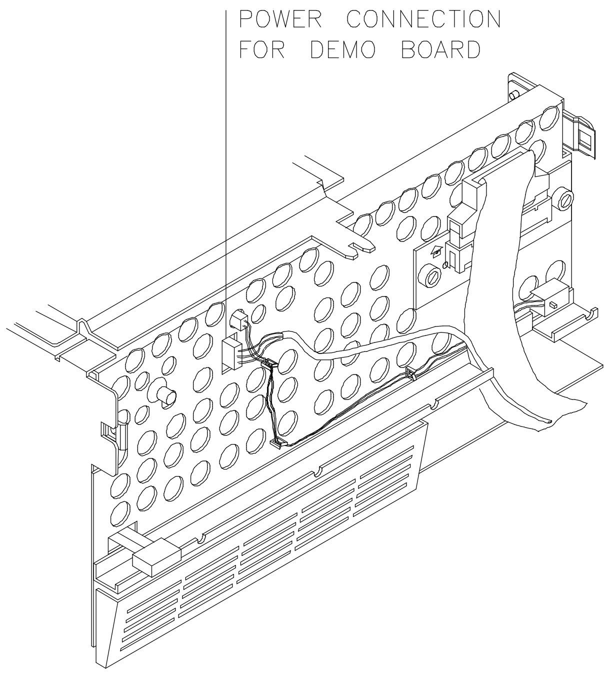
Do not remove the system control board. This board is used in all HP 64700 emulation and analysis systems.

10 Install HP 64704A and HP 64748C boards. The HP 64704A is installed in the slot next to the system controller board. The HP 64748C is installed in the second slot from the bottom of the HP 64700. These boards are identified with labels that show the model number and the serial number.

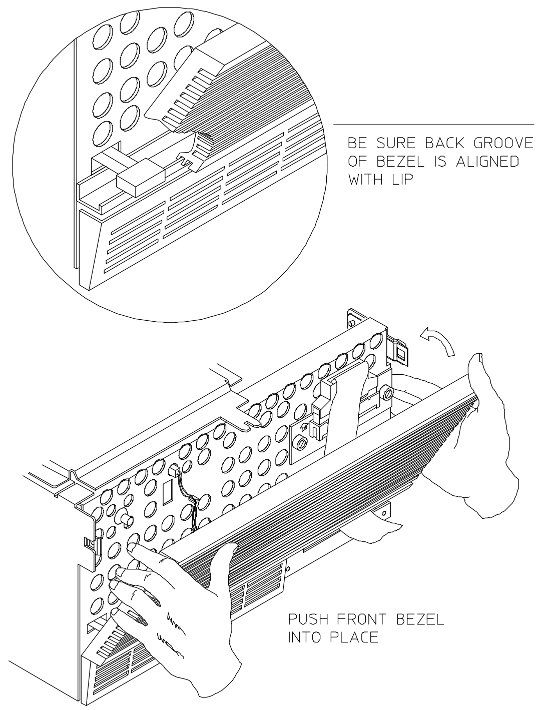
To install a card, insert it into the plastic guides. Make sure the connectors are properly aligned; then, press the card into mother board sockets. Check to ensure that the cards are seated all the way into the sockets. If the cards can be removed with your fingers, the cards are NOT seated all the way into the mother board socket.



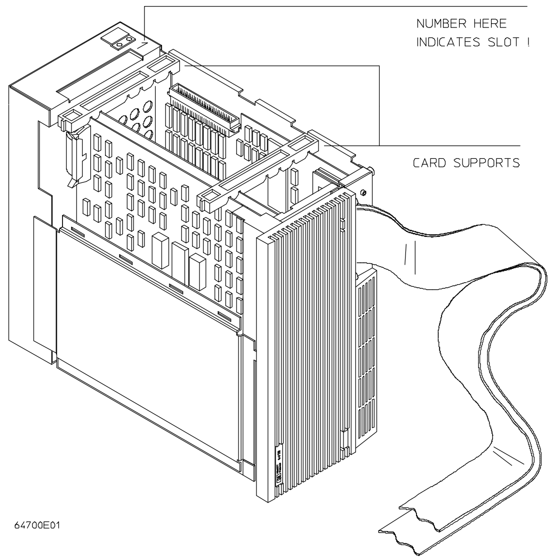
11 Connect the +5 V power cable to the connector in the HP 64700 front panel.



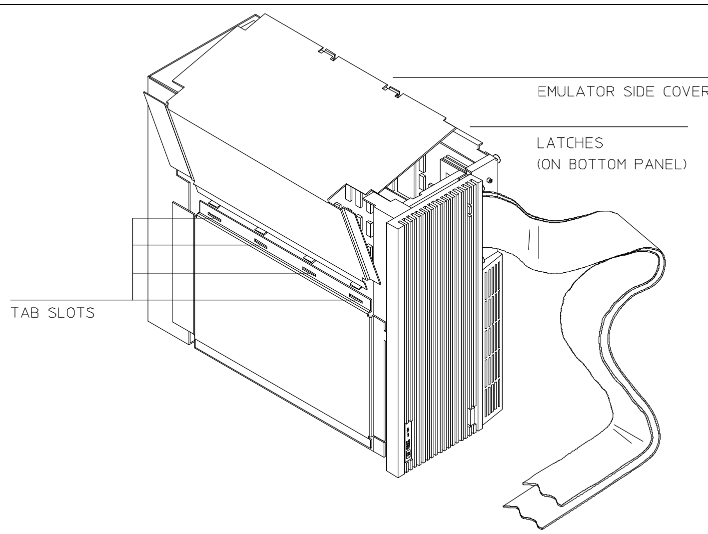
12 To reinstall the front bezel, be sure that the bottom rear groove of the front bezel is aligned with the lip as shown below.



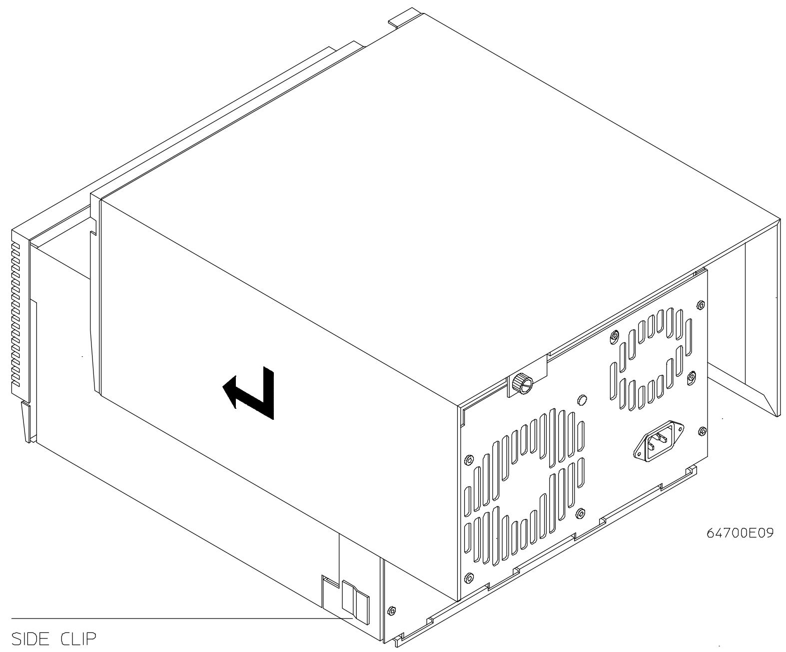
13 Install the card supports.



14 To install the side cover, insert the side cover into the tab slots and fasten the two latches.

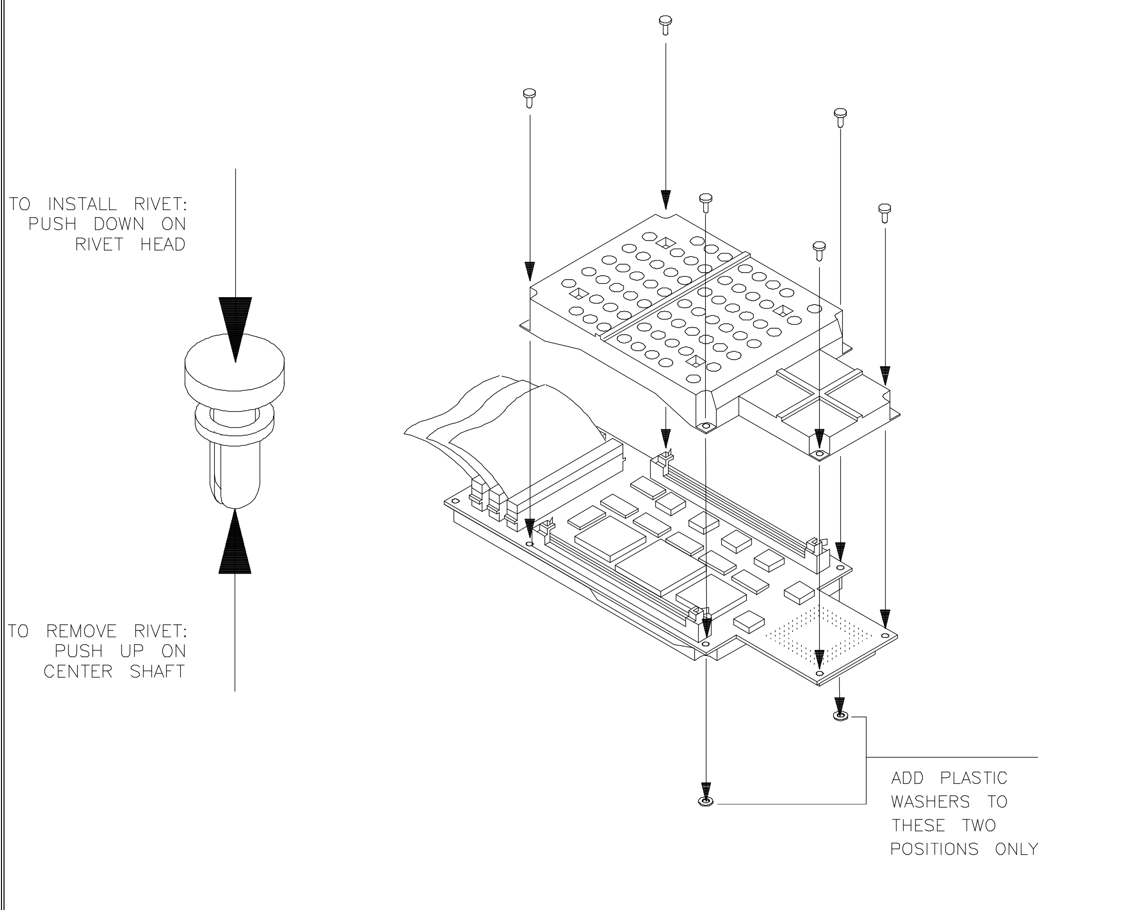


15 Install the top cover in reverse order of its removal, but make sure that the side panels of the top cover are attached to the side clips on the frame.



Step 3. Install emulation memory modules on emulator probe

1 Remove plastic rivets that secure the plastic cover on the top of the emulator probe, and remove the cover.



2 Determine the placement of the emulation memory modules. Three types of modules may be installed: 256 Kbyte (HP 64171A or HP 64172A), 1 Mbyte (HP 64171B or HP 64172B), and 4 Mbyte (HP 64173A). Any type of module may be installed in either bank.

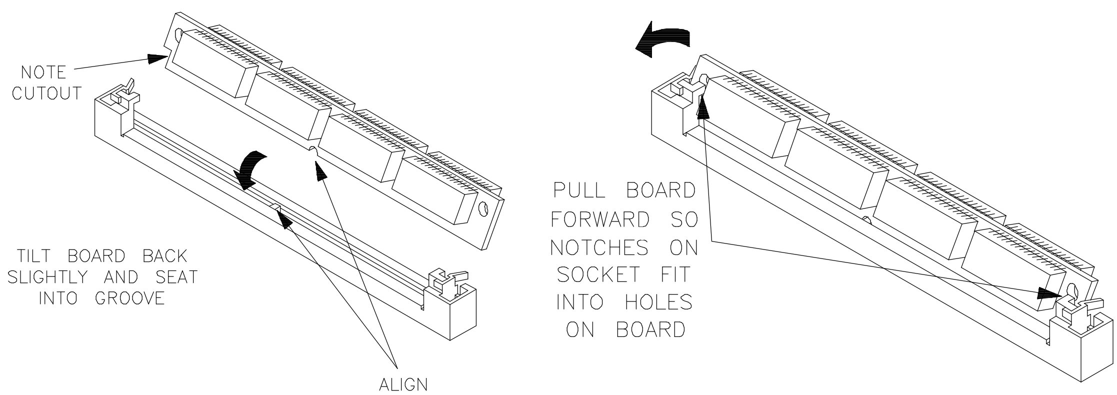
Memory in bank 0 is divided into 4 equal blocks that can be allocated by the memory mapper. Memory in bank 1 is divided into 2 equal blocks.

If you have only one emulation memory module, place it in bank 0 to give yourself greater flexibility when mapping address ranges to emulation memory. If you have two memory modules and one is larger than the other, place the larger module in bank 0 to give yourself more evenly proportioned blocks.

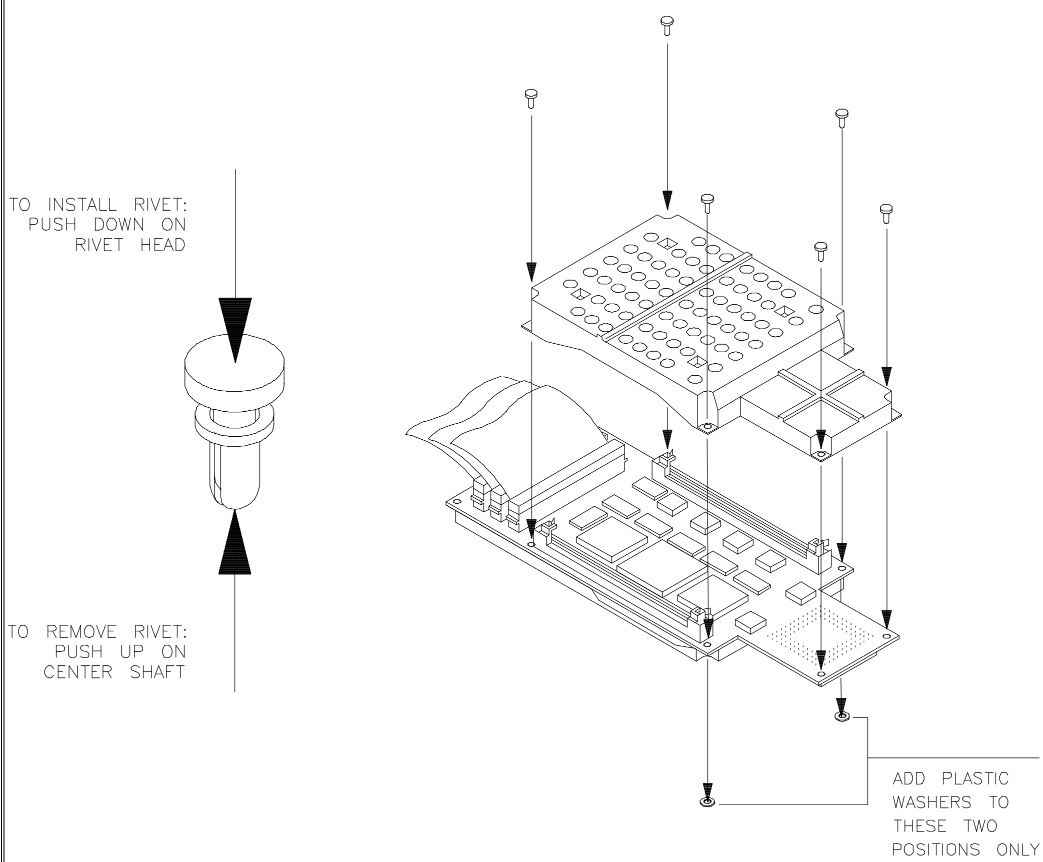
The HP 64171A/B memory modules provide 0 wait state emulation memory through 16.7 MHz and 1 wait state above 16.7 MHz. The HP 64172A/B memory modules provide 0 wait state emulation memory through 25 MHz. The HP 64173A memory modules provide 0 wait state emulation memory through 22 MHz and 1 wait state above 22 MHz. (The 68340 processor is programmed for the correct number of wait states by user code.) If memory modules are mixed, the performance characteristics of the slower module should be used.

3 Install emulation memory modules on emulator probe. There is a cutout on one side of the memory modules so that they can only be installed one way.

To install memory modules, place the memory module into the socket groove at an angle. Firmly press the memory module into the socket to make sure that it is completely seated. Once the memory module is seated in the connector groove, pull the memory module forward so that the notches on the socket fit into the holes on the memory module. There are two latches on the sides of the socket that hold the memory module in place.

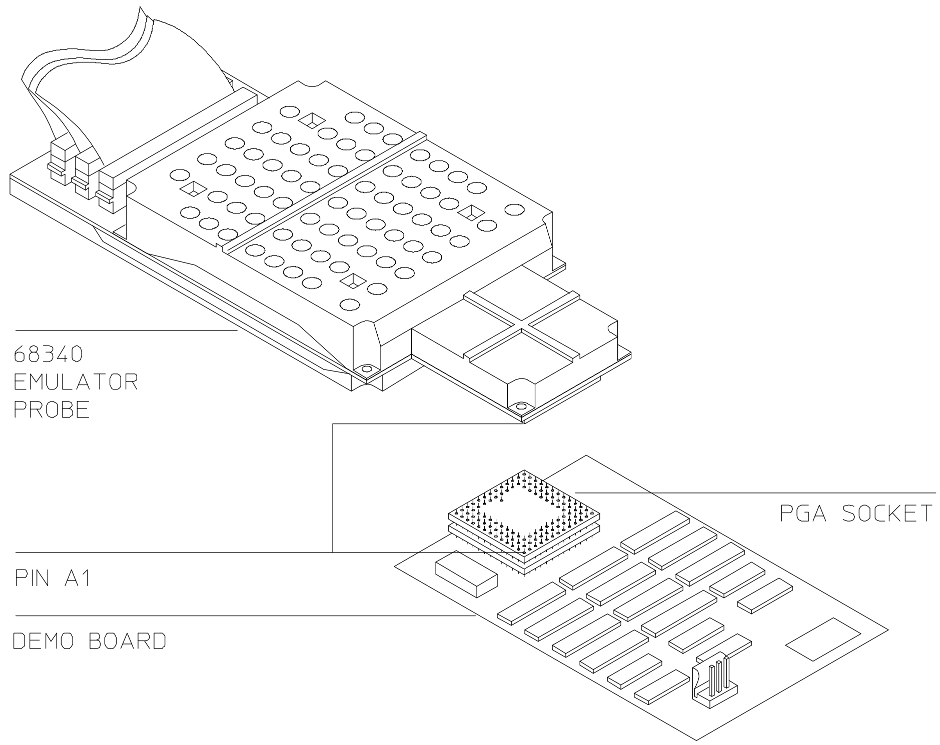


4 Replace the plastic cover, and insert new plastic rivets (supplied with the emulator) to secure the cover.

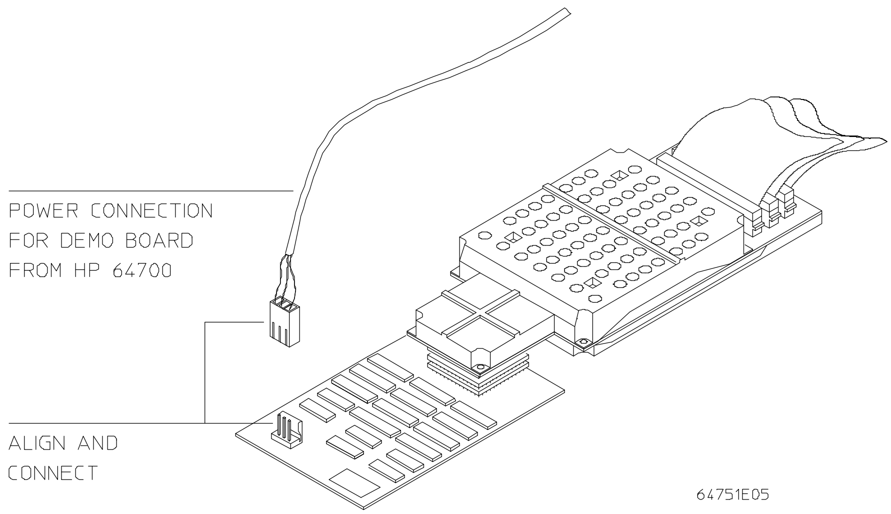


Step 4. Plug the emulator probe into the demo target system

1 With HP 64700 power OFF, connect the emulator probe cables to the demo target system.



2 Connect the power supply wires from the emulator to the demo target system. The 3-wire cable has 1 power wire and 2 ground wires. **When attaching the 3-wire cable to the demo target system, make sure the connector is aligned properly so that all three pins are connected**.



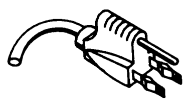
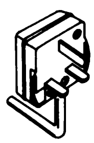
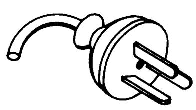
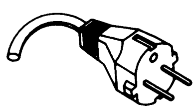
Step 5. Apply power to the HP 64700

The HP 64700B automatically selects the 115 Vac or 220 Vac range. In the 115 Vac range, the HP 64700B will draw a maximum of 345 W and 520 VA. In the 220 Vac range, the HP 64700B will draw a maximum of 335 W and 600 VA.

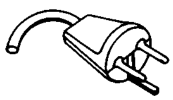
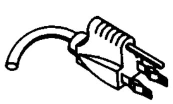
The HP 64700 is shipped from the factory with a power cord appropriate for your country. You should verify that you have the correct power cable for installation by comparing the power cord you received with the HP 64700 with the drawings under the "Plug Type" column of the following table.

If the cable you received is not appropriate for your electrical power outlet type, contact your Hewlett-Packard sales and service office.

Power Cord Configurations

Plug Type	Cable Part No.	Plug Description	Length in/cm	Color
Opt 903 124V ** 	8120-1378 8120-1521	Straight * NEMA5-15P 90^{o}	90/228 90/228	Jade Gray Jade Gray
Opt 900 250V 	8120-1351 8120-1703	Straight * BS136A 90^{o}	90/228 90/228	Gray Mint Gray
Opt 901 250V 	8120-1369 8120-0696	Straight * NZSS198/ASC 90^{o}	79/200 87/221	Gray Mint Gray
Opt 902 250V 	812001689 8120-1692 8120-2857	Straight * CEE7-Y11 90^{o} Straight (Shielded)	79/200 79/200 79/200	Mint Gray Mint Gray Coco Brown
* Part number shown for plug is industry identifier for plug only. Number shown for cable is HP part number for complete cable including plug. ** These cords are included in the CSA certification approval for the equipment.				

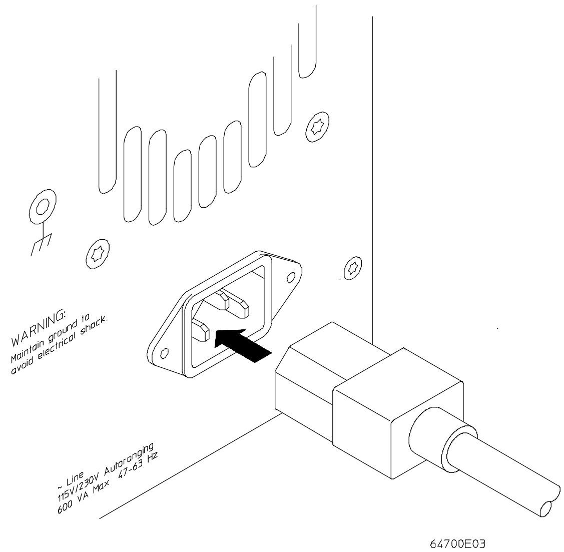
Power Cord Configurations (Cont'd)

Plug Type	Cable Part No.	Plug Description	Length in/cm	Color
Opt 906 250V 	8120-2104 8120-2296	Straight * SEV1011 1959-24507 Type 12 90^o	79/20 79/200	Mint Gray Mint Gray
Opt 912 220V 	 8120-2957	Straight *DHCK107 90^o	79/200 79/200	Mint Gray Mint Gray
Opt 917 250V 	8120-4600 8120-4211	Straight SABS164 90^o	79/200 79/200	Jade Gray
Opt 918 100V 	8120-4753 8120-4754	Straight Miti 90^o	90/230 90/230	Dark Gray

* Part number shown for plug is industry identifier for plug only.
Number shown for cable is HP part number for complete cable including plug.
** These cords are included in the CSA certification approval for the equipment.

1 Connect the power cord and turn on the HP 64700.

The line switch is a push button located at the lower left hand corner of the front panel. To turn ON power to the HP 64700, push the line switch button in to the ON (1) position. The power light at the lower right hand corner of the front panel will be illuminated.



Connecting the HP 64700 to a Computer or LAN

Refer to the *HP 64700 Series Installation/Service Guide* for instructions on connecting the HP 64700 to a host computer (via RS-422 or RS-232) or LAN and setting the HP 64700's configuration switches. (RS-422 and RS-232 are only supported on HP 9000 Series 300/400 machines.)

Installing HP 9000 Software

This section shows you how to install the Graphical User Interface on HP 9000 workstations. These instruction also tell you how not to install the Graphical User Interface if you want to use just the conventional Softkey Interface.

This section shows you how to:

1 Install the software from the media.

2 Verify the software installation.

3 Start the X server and the Motif Window Manager (mwm), or start HP VUE.

4 Set the necessary environment variables.

Step 1. Install the software from the media

The tape that contains the Graphical User Interface software may contain several products. Usually, you will want to install all of the products on the tape. However, to save disk space, or for other reasons, you can choose to install selected filesets.

If you plan on using the Softkey Interface instead of the Graphical User Interface, you can save about 3.5 megabytes of disk space by not installing the XUI suffixed filesets in the "64700 Operating Environment" and "<processor-type> Emulation Tools" partitions. (Also, if you choose not to install the Graphical User Interface, you will not have to use a special command line option to start the Softkey Interface.)

Refer to the information on updating HP-UX in your HP-UX documentation for instructions on viewing partitions and filesets and marking filesets that should not be loaded.

The following sub-steps assume that you want to install all products on the tape.

1 Become the root user on the system you want to update.

2 Make sure the tape's write-protect screw points to SAFE.

3 Put the product media into the tape drive that will be the *source device* for the update process.

4 Confirm that the tape drive BUSY and PROTECT lights are on. If the PROTECT light is not on, remove the tape and confirm the position of the write-protect screw. If the BUSY light is not on, check that the tape is installed correctly in the drive and that the drive is operating correctly.

5 When the BUSY light goes off and stays off, start the update program by entering

/etc/update

at the HP-UX prompt.

6 When the HP-UX update utility main screen appears, confirm that the source and destination devices are correct for your system. Refer to the information on updating HP-UX in your HP-UX documentation if you need to modify these values.

7 Select "Load Everything from Source Media" when your source and destination directories are correct.

8 To begin the update, press the softkey <Select Item>. At the next menu, press the softkey <Select Item> again. Answer the last prompt with

y

It takes about 20 minutes to read the tape.

9 When the installation is complete, read /tmp/update.log to see the results of the update.

Step 2. Verify the software installation

A number of new filesets were installed on your system during the software installation process. This and following steps assume that you chose to load the Graphical User Interface filesets.

You can use this step to further verify that the filesets necessary to successfully start the Graphical User Interface have been loaded and that customize scripts have run correctly. Of course, the update process gives you mechanisms for verifying installation, but these checks can help to double-check the install process.

1 Verify the existence of the **HP64_Softkey** file in the **/usr/lib/X11/app-defaults** subdirectory by entering
ls /usr/lib/X11/app-defaults/HP64_Softkey at the HP-UX prompt.

Finding this file verifies that you loaded the correct fileset and also verifies that the customize scripts executed because this file is created from other files during the customize process.

2 Examine **/usr/lib/X11/app-defaults/HP64_Softkey** near the end of the file to confirm that there are resources specific to your emulator.

Near the end of the file, there will be resource strings that contain references to specific emulators. For example, if you installed the Graphical User Interface for the 68340 emulator, resource name strings will have **m68340** embedded in them.

After you have verified the software installation, you must start the X server and an X window manager (if you are not currently running an X server). If you plan to run the Motif Window Manager (mwm), or similar window manager, continue with Step 3a of these instructions. If you plan to run HP VUE, skip to Step 3b of these instructions.

Step 3a. Start the X server and the Motif Window Manager (mwm)

If you are not already running the X server and a window manager, do so now. The X server is required to use the Graphical User Interface because it is an X Windows application. A window manager is not required to execute the interface, but, as a practical matter, you must use some sort of window manager with the X server.

- Start the X server by entering **x11start** at the HP-UX prompt.

Consult the X Window documentation supplied with the HP-UX operating system documentation if you do not know about using X Windows and the X server.

After starting the X server and Motif Window Manager, continue with step 4 of these instructions.

Step 3b. Start HP VUE

If you are running the X server under HP VUE and have not started HP VUE, do so now.

HP VUE is a window manager for the X Window system. The X server is executing underneath HP VUE. Unlike the Motif Window Manager, HP VUE provides a login shell and is your default interface to the HP 9000 workstation.

Step 4. Set the necessary environment variables

The DISPLAY environment variable must be set before the Graphical User Interface will start. Also, you should modify the PATH environment variable to include the "/usr/hp64000/bin" directory, and, if you have installed software in a directory other than "/", you need to set the HP64000 environment variable.

The following instructions show you how to set these variables at the UNIX prompt. Modify your ".profile" or ".login" file if you wish these environment variables to be set when you log in. The following instructions also assume that you're using "sh" or "ksh"; if you're using "csh", environment variables are set using the "setenv <VARIABLE> <value>" command.

1 Set the DISPLAY environment variable by entering

DISPLAY=<hostname>:<server_number>.<screen_number>
export DISPLAY

For example:

DISPLAY=myhost:0.0; export DISPLAY

Consult the X Window documentation supplied with the UNIX system documentation for an explanation of the DISPLAY environment variable.

2 Set the HP64000 environment variable.

For example, if you installed the HP 64000 software relative to the root directory, "/", you would enter

HP64000=/usr/hp64000; export HP64000

If you installed the software relative to a directory other than the root directory, it is strongly recommended that you use a symbolic link to make the software appear to be under /usr/hp64000. For example, if you installed the software relative to directory /users/team, you would enter

ln -s /users/team/usr/hp64000 /usr/hp64000

If you do not wish to establish a symbolic link, you can set the HP64000 variable to the full path that contains the HP 64000 software. Again, if you installed relative to /users/team, you would enter

HP64000=/users/team/usr/hp64000; export HP64000

3 Set the PATH environment variable to include the **usr/hp64000/bin** directory by entering

PATH=$PATH:$HP64000/bin; export PATH

Including **usr/hp64000/bin** in your PATH relieves you from prefixing HP 64700 executables with the directory path.

4 Set the MANPATH environment variable to include the **usr/hp64000/man** and **usr/hp64000/contrib/man** directories by entering

MANPATH=$MANPATH:$HP64000/man:$HP64000/contrib/man
export MANPATH

Including these directories in your MANPATH variable lets you access the on-line "man" page information included with the software.

Installing Sun SPARCsystem Software

This section shows you how to install the Graphical User Interface on Sun SPARCsystem workstations. These instructions also tell you how not to install the Graphical User Interface if you want to use just the conventional Softkey Interface.

This section shows you how to:

1 Install the software from the media.

2 Start the X server and OpenWindows.

3 Set the necessary environment variables.

4 Verify the software installation.

5 Map your function keys.

Step 1. Install the software from the media

The tape that contains the Graphical User Interface software may contain several products. Usually, you will want to install all of the products on the tape. However, to save disk space, or for other reasons, you can choose to install selected filesets.

If you plan on using the conventional Softkey Interface instead of the Graphical User Interface, you can save about 3.5 megabytes of disk space by not installing the XUI suffixed filesets. (Also, if you choose not to install the Graphical User Interface, you will not have to use a special command line option to start the Softkey Interface.)

Refer to the *Software Installation Notice* for software installation instructions. After you are done installing the software, return here.

Step 2. Start the X server and OpenWindows

If you are not already running the X server, do so now. The X server is required to run the Graphical User Interface because it is an X application.

- Start the X server by entering **/usr/openwin/bin/openwin** at the UNIX prompt.

Consult the OpenWindows documentation if you do not know about using OpenWindows and the X server.

Step 3. Set the necessary environment variables

The DISPLAY environment variable must be set before the Graphical User Interface will start. Also, you should modify the PATH environment variable to include the "usr/hp64000/bin" directory, and, if you have installed software in a directory other than "/", you need to set the HP64000 environment variable.

The following instructions show you how to set these variables at the UNIX prompt. Modify your ".profile" or ".login" file if you wish these environment variables to be set when you log in. The following instructions also assume that you're using "csh"; if you're using "sh", environment variables are set in the "<VARIABLE>=<value>; export <VARIABLE>" form.

1 The DISPLAY environment variable is usually set by the **openwin** startup script. Check to see that DISPLAY is set by entering

echo $DISPLAY

If DISPLAY is not set, you can set it by entering

setenv DISPLAY=<hostname>:<server_number>.<screen_number>

For example:

setenv DISPLAY=myhost:0.0

Consult the OpenWindows documentation for an explanation of the DISPLAY environment variable.

2 Set the HP64000 environment variable.

For example, if you installed the HP 64000 software relative to the root directory, "/", you would enter

setenv HP64000 /usr/hp64000

If you installed the software relative to a directory other than the root directory, it is strongly recommended that you use a symbolic link to make the software appear to be under /usr/hp64000. For example, if you installed the software relative to directory /users/team, you would enter

ln -s /users/team/usr/hp64000 /usr/hp64000

If you do not wish to establish a symbolic link, you can set the HP64000 variable to the full path that contains the HP 64000 software; also set the LD_LIBRARY_PATH variable to the directory containing run-time libraries used by the HP 64000 products. Again, if you installed relative to /users/team, you would enter

setenv HP64000 /users/team/usr/hp64000
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${HP64000}/lib

3 Set the PATH environment variable to include the **usr/hp64000/bin** directory by entering

setenv PATH ${PATH}:${HP64000}/bin

Including **usr/hp64000/bin** in your PATH relieves you from prefixing HP 64700 executables with the directory path.

4 Set the MANPATH environment variable to include the **usr/hp64000/man** and **usr/hp64000/contrib/man** directories by entering

setenv MANPATH ${MANPATH}:${HP64000}/man
setenv MANPATH ${MANPATH}:${HP64000}/contrib/man

Including these directories in your MANPATH variable lets you access the on-line "man" page information included with the software.

5 If the Graphical User Interface is to run on a SPARCsystem computer that is not running OpenWindows, include the /usr/openwin/lib directory in LD_LIBRARY_PATH.

setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:/usr/openwin/lib

Step 4. Verify the software installation

A number of product filesets were installed on your system during the software installation process. Due to the complexity of installing on NFS mounted file systems, a script that verifies and customizes these products was also installed. This stand alone script may be run at any time to verify that all files required by the products are in place in the file system. If required files are not found, this script will attempt to symbolically link them from the $HP64000 install directory to their proper locations.

- Run the script **$HP64000/bin/envinstall**.

Step 5. Map your function keys

If you are using the conventional Softkey Interface, map your function keys by following the steps below.

1 Copy the function key definitions by typing:

cp $HP64000/etc/ttyswrc ~/.ttyswrc

This creates key mappings in the .ttyswrc file in your $HOME directory.

2 Remove or comment out the following line from your .xinitrc file:

```
xmodmap -e 'keysym F1 = Help'
```

If any of the other keys F1-F8 are remapped using xmodmap, comment out those lines also.

3 Add the following to your .profile or .login file:

```
stty erase ^H
setenv KEYMAP sun
```

The erase character needs to be set to backspace so that the Delete key can be used for "delete character."

If you want to continue using the F1 key for HELP, you can use use F2-F9 for the Softkey Interface. All you have to do is set the KEYMAP variable. If you use OpenWindows, type:

setenv KEYMAP sun.2-9

If you use xterm windows (the xterm window program is located in the directory /usr/openwin/demo), type:

setenv KEYMAP xterm.2-9

Reminder: If you are using OpenWindows, add /usr/openwin/bin to the end of the $PATH definition, and add the following line to your .profile:

```
setenv OPENWINHOME /usr/openwin
```

After you have mapped your function keys, you must start the X server and an X window manager (if you are not currently running an X server).

Verifying the Installation

This section shows you how to:

- Determine the logical name of your emulator.
- Start the emulator/analyzer interface for the first time.
- Exit the emulator/analyzer interface.

Step 1. Determine the logical name of your emulator

The *logical name* of an emulator is a label associated with a set of communications parameters in the **$HP64000/etc/64700tab.net** file. The 64700tab.net file is placed in the directory as part of the installation process.

1 Display the 64700tab.net file by entering
more /usr/hp64700/etc/64700tab.net at the HP-UX prompt.

2 Page through the file until you find the emulator you are going to use.

This step will require some matching of information to an emulator, but it should not be difficult to determine which emulator you want to address.

Examples

A typical entry for a 68340 emulator connected to the LAN would appear as follows:

```
#---------------------------------------------------------------------------
# Channel| Logical   | Processor | Remainder of Information for the Channel
#  Type  |  Name     |   Type    | (IP address for LAN connections)
#---------------------------------------------------------------------------
   lan:     em68340     m68340      21.17.9.143
```

A typical entry for a 68340 emulator connected to an RS-422 port would appear as follows:

```
#-------------------------------------------------------------------------------
#        |            |           |      |              |Xpar|Parity|Flow|Stop|Char
# Channel|  Logical   | Processor | Host |   Physical   |Mode|      |    |Bits|Size
#  Type  |   Name     |   Type    | Name |    Device    |    |      |XON |    |
#        |            |           |      |              |OFF | NONE |RTS | 2  | 8
#-------------------------------------------------------------------------------
  serial:   em68340     m68340    myhost /dev/emcom23 OFF   NONE  RTS   2    8
```

Step 2. Start the interface with the emul700 command

1 Apply power to the emulator you wish to access after making sure the emulator is connected to the LAN or to your host system.

On the HP 64700 Series Emulator, the power switch is located on the front panel near the bottom edge. Push the switch in to turn power on to the emulator.

2 Wait a few seconds to allow the emulator to complete its startup initialization.

3 Choose a terminal window from which to start the Graphical User Interface.

4 Start the Graphical User Interface by entering **emul700** command and giving the logical name of the emulator as an argument to the command, as in

$HP64000/bin/emul700 <logical_name> &

or
emul700 <logical name> &

if **$HP64000/bin** is in your path.

If you are running the X server, if the Graphical User Interface is installed, and if your DISPLAY environment variable is set, the **emul700** command will start the

Graphical User Interface. Otherwise, **emul700** starts the conventional Softkey Interface.

You should include an ampersand ("&") with the command to start the Graphical User Interface as a background process. Doing so frees the terminal window where you started the interface so that the window may still be used.

5 Optionally start additional Graphical User Interface windows into the same emulation session by repeating the previous step.

You can also choose to use the conventional Softkey Interface under X Windows, but you must include a command line argument to **emul700** to override the default Graphical User Interface. Start the conventional interface by entering

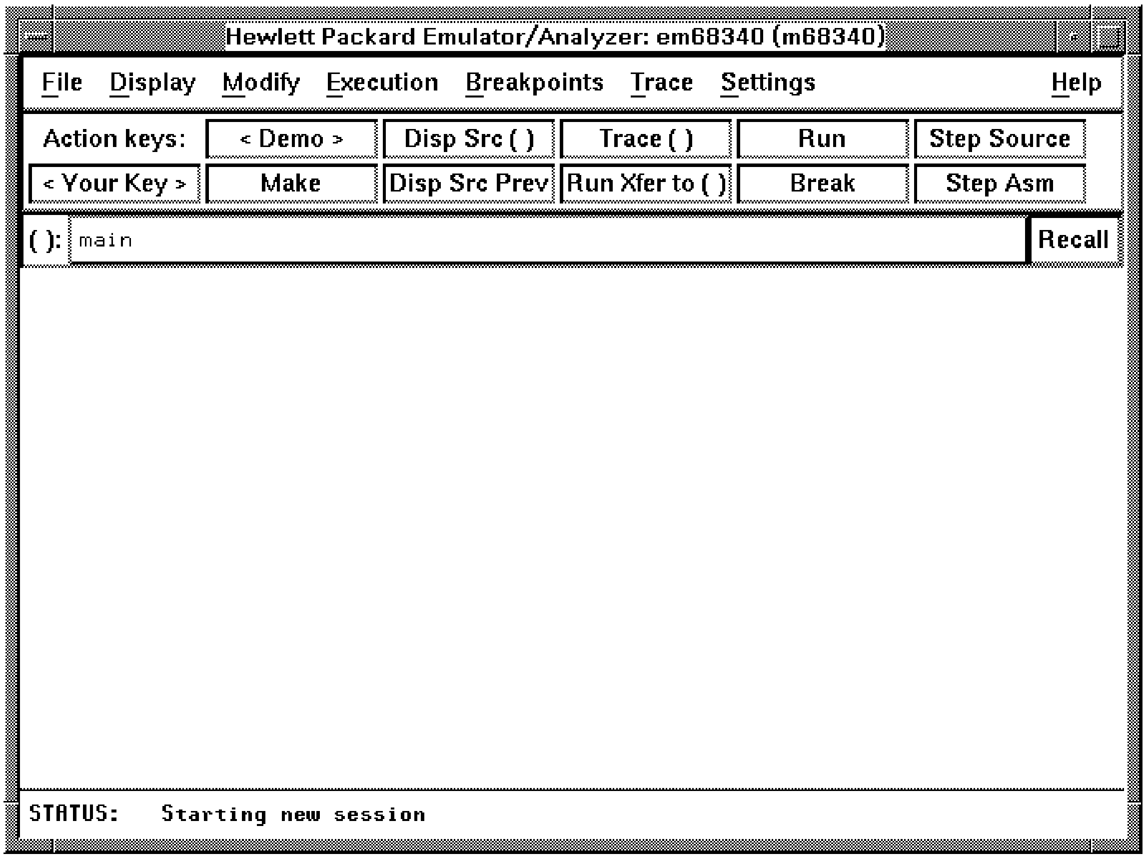
emul700 -u skemul <logical name>

Example

Suppose you have discovered that the logical name for a 68340 emulator connected to the LAN is "em68340". To start the Graphical User Interface and begin communicating with that emulator, enter (assuming your $PATH includes **$HP64000/bin**)

```
emul700 em68340
```

After a few seconds, the Graphical User Interface Emulator/Analyzer window should appear on your screen. The window will be similar to the following:


Hewlett Packard Emulator/Analyzer: em68340 (m68340)
File
Display
Modify
Execution
Breakpoints
Trace
Settings
Help
Action keys:
< Demo >
Disp Src ( )
Trace ( )
Run
Step Source
< Your Key >
Make
Disp Src Prev
Run Xfer to ( )
Break
Step Asm
( ):
main
Recall
STATUS: Starting new session

Step 3. Exit the Graphical User Interface

1 Position the mouse pointer over the pulldown menu named "File" on the menu bar at the top of the interface screen.

2 Press and hold the command select mouse button until the File menu appears.

3 While continuing to hold the mouse button down, move the mouse pointer down the menu to the "Exit" menu item.

4 Display the Exit cascade menu by moving the mouse pointer to the right edge of the Exit menu choice. There is an arrow on the right edge of the menu item.

5 Choose "Released" from the cascade menu.

The interface will terminate and release the emulator for use by others.

16

Installing/Updating Emulator Firmware

Installing/Updating Emulator Firmware

If you ordered the HP 64751A 68340 emulator probe and the HP 64748C emulation control card together, the control card contains the correct firmware for the HP 64751A.

However, if you ordered the HP 64751A and the HP 64748C separately, or if you are using a HP 64748C that has been used previously with a different emulator probe, you must download the correct firmware into the emulation control card.

The 68340 emulator firmware is included with the emulator/analyzer interface software, and the program that downloads emulator firmware is included with the HP B1471 64700 Operating Environment product.

(The firmware, and the program that downloads it into the control card, are also included with the 68340 emulator probe on an MS-DOS format floppies. The floppies are for users that do not have hosted interface software.)

Before you can update emulator firmware, you must have already installed the emulator into the HP 64700, connected the HP 64700 to a host computer or LAN, and installed the emulator/analyzer interface and HP B1471 software as described in the "Installation" chapter.

This chapter describes how to:

- Update firmware with the "progflash" command.
- Display current firmware version information.

To update emulator firmware with "progflash"

- Enter the **progflash -v <emul_name> <products ...>** command.

The **progflash** command downloads code from files on the host computer into Flash EPROM memory in the HP 64700.

The **-v** option means "verbose". It causes progress status messages to be displayed during operation.

The <emul_name> option is the logical emulator name as specified in the /usr/hp64000/etc/64700tab.net file.

The <products> option names the products whose firmware is to be updated.

If you enter the **progflash** command without options, it becomes interactive. If you don't include the <emul_name> option, it displays the logical names in the /usr/hp64000/etc/64700tab.net file and asks you to choose one. If you don't include the <products> option, it displays the products which have firmware update files on the system and asks you to choose one. (In the interactive mode, only one product at a time can be updated.) You can abort the interactive **progflash** command by pressing <CTRL>c.

progflash will print "Flash programming SUCCEEDED" and return 0 if it is successful; otherwise, it will print "Flash programming FAILED" and return a nonzero (error).

You can verify the update by displaying the firmware version information.

Examples

To update the "em68340" emulator firmware:

```
$ progflash <RETURN>
```

```
                HPB1471-19309 A.05.00 03Jan94
              64700 SERIES EMULATION COMMON FILES

               A Hewlett-Packard Software Product
               Copyright Hewlett-Packard Co. 1988


   All Rights Reserved. Reproduction, adaptation, or translation without prior
   written  permission  is prohibited, except as allowed under copyright laws.

                            RESTRICTED RIGHTS LEGEND

      Use , duplication , or disclosure  by the  Government is  subject to
      restrictions as set forth in subparagraph (c) (1) (II) of the Rights
      in Technical Data and Computer Software clause at  DFARS 52.227-7013.
      HEWLETT-PACKARD Company , 3000 Hanover St. , Palo Alto, CA 94304-1181

             Logical Name        Processor
           1 em68k               m68000
           2 em80960             i80960
           3 em68340             m68340

 Number of Emulator to Update? (intr (usually cntl C or DEL) to abort)
```

To update firmware in the HP 64700 that contains the 68340 emulator, enter "3".

```
             Product
           1 64700
           2 64703/64704/64706/64740
           3 64744
           4 64760
           5 64751

 Number of Product to Update? (intr (usually cntl C or DEL) to abort)
```

To update the HP 64751A 68340 emulator firmware, enter "5".

```
 Enable progress messages? [y/n] (y)
```

To enable status messages, enter "y".

```
Checking System firmware revision...
Mainframe is a 64700B

Reading configuration from '/usr/hp64000/inst/update/64751.cfg'
ROM identifier address = 2FFFF0H
Required hardware identifier = 1FF4H
Control ROM start address = 280000H
Control ROM size = 40000H
Control ROM width = 16
Programming voltage control address = 2FFFFEH
Programming voltage control value = FFFFH
Programming voltage control mask = 0H

Rebooting HP64700...
Checking Hardware id code...
Erasing Flash ROM
Downloading ROM code: /usr/hp64000/inst/update/64751.X
    Code start 280000H (should equal control ROM start)
    Code size 2348CH (must be less than control ROM size)
Finishing up...

Rebooting HP64700...
Flash programming SUCCEEDED
$
```

You could perform the same update as in the previous example with the following command:

$ **progflash -v** em68340 64751 <RETURN>

To display current firmware version information

- Use the Terminal Interface **ver** command to view the version information for firmware currently in the HP 64700.

  When using the Graphical User Interface or Softkey Interface, you can enter Terminal Interface commands with the **pod_command** command. For example:

  display pod_command <RETURN>
  pod_command "ver" <RETURN>

Examples

The Terminal Interface **ver** command displays information similar to:

```
                  Copyright (c) Hewlett-Packard Co. 1987
All Rights Reserved.  Reproduction, adaptation, or translation without prior
written permission is prohibited, except as allowed under copyright laws.

  HP64700B Series Emulation System
    Version:   B.01.00 20Dec93
    Location:  Flash
    System RAM:1 Mbyte

  HP64751A Motorola 68340 Emulator
    Version:   A.00.00 26Mar92
    Control:   HP64748C  Emulation Control Board
    Speed:     16.7 MHz
    Memory:    260 Kbytes
      Bank 0:  HP64171A 256 Kbyte 35ns Memory Module

  HP64740 Emulation Analyzer
    Version:   A.02.02 13Mar91
```

If there is a power failure during a firmware update

If there is a power glitch during a firmware update, some bits may be lost during the download process, possibly resulting in an HP 64700 that will not boot up.

☐ Repeat the firmware update process.

☐ If the HP 64700 is connected to the LAN in this situation and you are unable to connect to the HP 64700 after the power glitch, try repeating the firmware update with the HP 64700 connected to an RS-232 or RS-422 interface.

Glossary

access mode Specifies the types of cycles used to access target system memory locations. For example a "byte" access mode tells the monitor program to use load/store byte instructions to access target memory.

analyzer An instrument that captures data on signals of interest at discreet periods.

background The emulator mode in which foreground operation is suspended so the emulation processor can be used for communication with the emulation controller. The background monitor does not occupy any processor address space.

background emulation monitor An emulation monitor that does not execute as part of the user program, and therefore, operates in the emulator's background mode.

display mode When displaying memory, this mode tells the emulator the size of the memory locations to display. When modifying memory, the display mode tells the emulator the size of the values to be written to memory.

embedded microprocessor system The microprocessor system which the emulator plugs into.

emulation bus analyzer The internal analyzer that captures emulator bus cycle information synchronously with the processor's clock signal.

emulation monitor program A program that is executed by the emulation processor which allows the emulation controller to access target system resources. For example, when you display target system memory locations, the monitor program executes microprocessor instructions that read the target memory locations and send their contents to the emulation controller.

emulator An instrument that performs just like the microprocessor it replaces, but at the same time, it gives you information about the operation of the processor. An emulator gives you control over target system execution and allows you to view or

modify the contents of processor registers, target system memory, and I/O resources.

foreground The mode in which the emulator is executing the user program. In other words, the mode in which the emulator operates as the target microprocessor would.

global restart When the same secondary branch condition is used for all terms in the analyzer's sequencer, and secondary branches are always back to the first term.

prestore The analyzer feature that allows up to two states to be stored before normally stored states. This feature is useful when you want to find the cause of a particular state. For example, if a variable is accessed from many different places in the program, you can qualify the trace so that only accesses of that variable are stored and turn on prestore to find out where accesses of that variable originate from.

primary sequencer branch Occurs when the analyzer finds the primary branch state specified at a certain level and begins searching for the states specified at the primary branch's destination level.

real-time Refers to continuous execution of the user program without interference from the emulator. (Such interference occurs when the emulator temporarily breaks into the monitor so that it can access register contents or target system memory or I/O.)

secondary sequencer branch Occurs when the analyzer finds the secondary branch state specified at a certain level before it found the primary branch state and begins searching for the states specified at the secondary branch's destination level.

sequence terms Individual levels of the sequencer. The analyzer provides 8 sequence terms.

sequencer The part of the analyzer that allows it to search for a certain sequence of states before triggering.

sequencer branch Occurs when the analyzer finds the primary or secondary branch state specified at a certain level and begins searching for the states specified at another level.

target system The microprocessor system which the emulator plugs into.

trace A collection of states captured on the emulation bus (in terms of the emulation bus analyzer) or on the analyzer trace signals (in terms of the external analyzer) and stored in trace memory.

trigger The captured analyzer state about which other captured states are stored. The trigger state specifies when the trace measurement is taken.

Index

Certification and Warranty

Certification

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.

Warranty

This Hewlett-Packard system product is warranted against defects in materials and workmanship for a period of 90 days from date of installation. During the warranty period, HP will, at its option, either repair or replace products which prove to be defective.

Warranty service of this product will be performed at Buyer's facility at no charge within HP service travel areas. Outside HP service travel areas, warranty service will be performed at Buyer's facility only upon HP's prior agreement and Buyer shall pay HP's round trip travel expenses. In all other cases, products must be returned to a service facility designated by HP.

For products returned to HP for warranty service, Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country. HP warrants that its software and firmware designated by HP for use with an instrument will execute its programming instructions when properly installed on that instrument. HP does not warrant that the operation of the instrument, or software, or firmware will be uninterrupted or error free.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environment specifications for the product, or improper site preparation or maintenance.

No other warranty is expressed or implied. HP specifically disclaims the implied warranties of merchantability and fitness for a particular purpose.

Exclusive Remedies

The remedies provided herein are buyer's sole and exclusive remedies. HP shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.

Safety

Summary of Safe Procedures

The following general safety precautions must be observed during all phases of operation, service, and repair of this instrument. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the instrument. Hewlett-Packard Company assumes no liability for the customer's failure to comply with these requirements.

Ground The Instrument

To minimize shock hazard, the instrument chassis and cabinet must be connected to an electrical ground. The instrument is equipped with a three-conductor ac power cable. The power cable must either be plugged into an approved three-contact electrical outlet or used with a three-contact to two-contact adapter with the grounding wire (green) firmly connected to an electrical ground (safety ground) at the power outlet. The power jack and mating plug of the power cable meet International Electrotechnical Commission (IEC) safety standards.

Do Not Operate In An Explosive Atmosphere

Do not operate the instrument in the presence of flammable gases or fumes. Operation of any electrical instrument in such an environment constitutes a definite safety hazard.

Keep Away From Live Circuits

Operating personnel must not remove instrument covers. Component replacement and internal adjustments must be made by qualified maintenance personnel. Do not replace components with the power cable connected. Under certain conditions, dangerous voltages may exist even with the power cable removed. To avoid injuries, always disconnect power and discharge circuits before touching them.

Do Not Service Or Adjust Alone

Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

Do Not Substitute Parts Or Modify Instrument

Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification of the instrument. Return the instrument to a Hewlett-Packard Sales and Service Office for service and repair to ensure that safety features are maintained.

Dangerous Procedure Warnings

Warnings, such as the example below, precede potentially dangerous procedures throughout this manual. Instructions contained in the warnings must be followed.

WARNING Dangerous voltages, capable of causing death, are present in this instrument. Use extreme caution when handling, testing, and adjusting.

Safety Symbols Used In Manuals

The following is a list of general definitions of safety symbols used on equipment or in manuals:

Instruction manual symbol: the product is marked with this symbol when it is necessary for the user to refer to the instruction manual in order to protect against damage to the instrument.

Indicates dangerous voltage (terminals fed from the interior by voltage exceeding 1000 volts must be marked with this symbol).

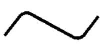
 OR 

Protective conductor terminal. For protection against electrical shock in case of a fault. Used with field wiring terminals to indicate the terminal which must be connected to ground before operating the equipment.

Low-noise or noiseless, clean ground (earth) terminal. Used for a signal common, as well as providing protection against electrical shock in case of a fault. A terminal marked with this symbol must be connected to ground in the manner described in the installation (operating) manual before operating the equipment.

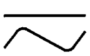
 OR 

Frame or chassis terminal. A connection to the frame (chassis) of the equipment which normally includes all exposed metal structures.



Alternating current (power line).

Direct current (power line).



Alternating or direct current (power line).

Caution

The Caution sign denotes a hazard. It calls your attention to an operating procedure, practice, condition, or similar situation, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product.

Warning

The Warning sign denotes a hazard. It calls your attention to a procedure, practice, condition or the like, which, if not correctly performed, could result in injury or death to personnel.